IMPLEMENTATION OF A LOW COST RECONFIGURABLE TRANSFORM ARCHITECTURE FOR MULTIPLE VIDEO CODECS

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

In the Department of Electrical and Computer Engineering

University of Saskatchewan

Saskatoon, Saskatchewan, Canada

By

MUHAMMAD ALI MARTUZA

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Electrical and Computer Engineering

57 Campus Drive

University of Saskatchewan

Saskatoon, Saskatchewan

Canada, S7N 5A9

ABSTRACT

Currently different types of transform techniques are used by different video codecs to achieve data compression during video frame transmission. Among them, Discrete Cosine Transform (DCT) is supported by most of modern video standards. The integer DCT (Int-DCT) is an integer approximation of DCT. It can be implemented exclusively with integer arithmetic. Int-DCT proves to be highly advantageous in cost and speed for hardware implementations. In particular, the 4x4 and 8x8 block size Int-DCTs have the increased applicability at the current multimedia industry because of their simpler implementation and better de-correlation performance for high definition (HD) video signals.

In this thesis, we present a fast and cost-shared reconfigurable architecture to compute variable block size Int-DCT for four modern video codecs – AVS, H.264/AVC, VC-1 and HEVC (under development). Based on the symmetric structure of the transform matrices and the similarity in matrix operations, we have developed a generalized "decompose and share" algorithm to compute the 4x4 and 8x8 block size Int-DCT. The algorithm is later applied to those four video codecs. Our shared hardware approach ensures the maximum circuit reuse during the computation. The entire architecture is multiplier free and designed with only adders and shifters to minimize hardware cost and improve working frequency.

Finally, the design is implemented on a FPGA and later synthesized in CMOS 0.18um technology to compare the cost and performance with existing designs. The results show significant reduction in hardware cost and meet the requirements of real time video coding applications.

ACKNOWLEDGMENTS

The work presented in this thesis has been carried out at Department of Electrical and Computer Engineering, University of Saskatchewan, Saskatoon, Saskatchewan during the years 2010 - 2012 under the kind supervision of Dr. Khan A. Wahid.

First, I would like to express my gratitude to my supervisor, Dr. Wahid for introducing me to this research field and for his continuous support, inspiration, and guidance throughout the project. His vast knowledge and expertise in this field added considerably to my graduate experience. I am also grateful to my co-supervisor, Prof. McCrosky for the financial support and guidance.

Then, I would like to thank my thesis committee for their insightful comments. I would also like to acknowledge the financial support from NSERC. Finally, I would like to thank my family for their love, encouragement, and support.

- Muhammad Ali Martuza

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
ASIC	Application-Specific Integrated Circuit
AVC	Advance Video Codec
AVS	Audio Video coding Standard
CMOS	Complementary-symmetry Metal Oxide Semiconductor
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
FPGA	Field Programmable Gate Array
fps	Frame Per Second
HD	High Definition
HDL	Hardware Description Language
HDTV	High Definition Television
HEVC	High Efficiency Video Coding
IDCT	Inverse Discrete Cosine Transform
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
Int-DCT	Integer Discrete Cosine Transform
Int-IDCT	Integer Inverse Discrete Cosine Transform
ISO	International Organization for Standardization

ITU	International Telecommunication Union
JCT-VC	Joint Collaborative Team on Video Coding
LUT	Lookup Table
MC	Motion Compensation
MPEG	Moving Picture Experts Group
msec	Milli-second
mW	Milli-watt
SMPTE	Society of Motion Picture and Television Engineers
SOC	System On a Chip
VC-1	Video Codec -1
VCEG	Video Coding Experts Group
VLSI	Very Large Scale Integration
WMV9	Windows Media Video 9
WQXGA	Wide Quad eXtended Graphics Array

CHAPTER 1

INTRODUCTION

1.1 Introduction to video compression system

This section presents a brief overview of the subject materials that are related to this research work. It begains with a simple definition of video signals and video compression systems. Then the importance of a transform unit inside a video compression system is described. Finally, different types of modern transform techniques are briefly described.

1.1.1 Video signal

A video signal is a series of image frames. It simulates a scene of motion by changing those frames in a sequence. In Figure 1.1, an example of video frame formation is shown.

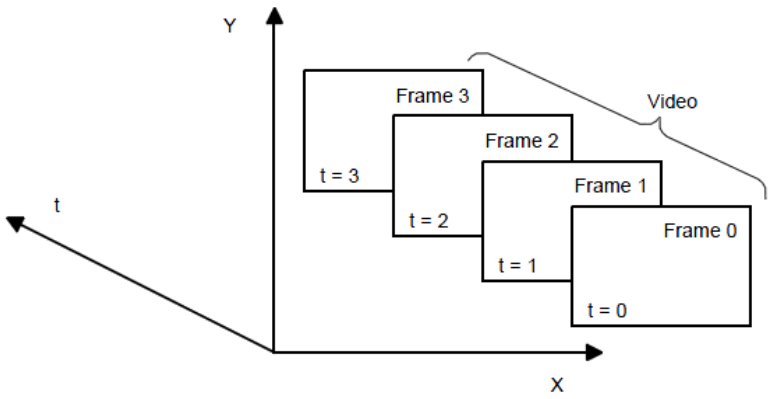

Figure 1.1. Example of video frame formation [1]

Here the ‘X’ and ‘Y’ axes represent the dimensions of the image frame and the ‘t’ axis represents time.

1.1.2 Video compression

Video compression is one of the key steps in a modern multimedia system. The size of the raw video frames is so large that without compression, the transmission process will be very expensive. That is why compression is necessary. It is the process of reducing the amount of data required to represent a given quantity of image frames. Mathematically, video compression is a process of transforming the pixel array of an image into a statistically uncorrelated data set. This compression is essentially a technique to reduce redundancy. In a video sequence, adjacent frames are generally correlated. This kind of correlation is called temporal redundancy. An example of temporal redundancy is shown in Figure 1.2. The figure shows three consecutive frames of a video sequence 'Trevor'. All of the frames are the same except for the position of the hands and lips of the person. Hence, video compression systems reduce this temporal redundancy to achieve significant compression.

Frame 1 Frame 2 Frame 3

Figure 1.2. Sequence of video frame 'Trevor' [57]

The entire video compression system is a complex process. Figure 1.3 shows the fundamental blocks of the VC-1 based video compression encoder [33]. Inside a decoder, the inverse functions are performed. All modern video compression systems function more-or-less as illustrated in this diagram.

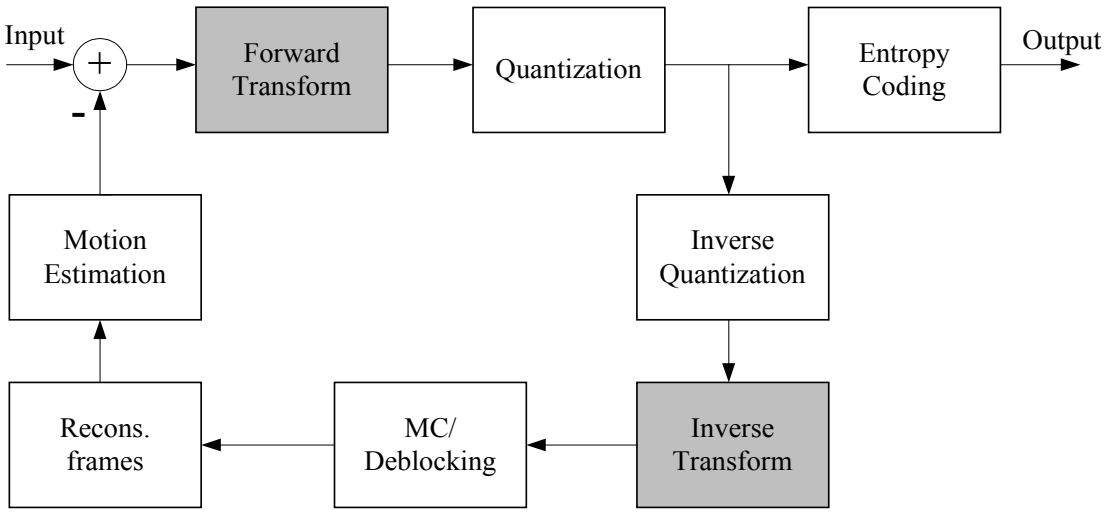

Figure 1.3. Block diagram of VC-1 based video compression encoder [33]

The compression techniques are mainly classified into two groups as follows:

I. Lossless compression

II. Lossy compression

I. Lossless compression

In a lossless data compression technique, the original data can be exactly reconstructed by the inverse process. This type of compression technique is generally used where the reconstruction quality is of utmost importance, such as, executable programs, text documents, source codes, and medical imaging.

II. Lossy compression

Lossy compression technique achieves data compression by losing some information while maintaining acceptable reconstruction quality. Hence, the data cannot be reconstructed exactly. This technique is used for applications where low storage space and fast data transmission speed are needed while maintaining the acceptable reconstructed data quality. The examples of such applications are still image compression, video conferencing, and internet telephony.

1.1.3 Transform coding in video compression system

A video compression system consists of different independent units like deblocking filters, transform, quantization, motion estimation, and motion compensation (as illustrated by Figure 1.3). Among them the transform unit is considered to be one of the major components of the compression system as it can reduce data considerably by performing transform coding.

The transformation of the data also makes the coefficients of the transformed matrix uncorrelated to each other. Image transforms operate directly on the pixels of the input image in the spatial domain. A 2D linear transform can be expressed in generalized form [2]:

$$T(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot r(x,y,u,v) \tag{1.1}$$

Here,

$f(x,y)$ = Input image frame (pixel matrix)
$r(x,y,u,v)$ = Basis function (forward transform kernel)
M, N = Row, Column dimension of f
u, v = Transform variable

1.1.4 Different types of transform

There are various transform coding methods used for video compression. Among them, the most popular transforms are listed below [3]:

I. Karhunen-Loeve Transform (KLT)

II. Discrete Fourier Transform (DFT)

III. Discrete Sine Transform (DST)

IV. Walsh Hadamard Transform (WHT)

V. Discrete Cosine Transform (DCT)

VI. Discrete Wavelet Transform (DWT)

A brief description of these techniques is presented here:

Karhunen-Loeve Transform: Among all the transformation techniques, the KLT is the most optimal transform which produces uncorrelated transformed coefficients [4]. Moreover, the KLT has the minimum mean square error as compared to other techniques [5]. It also has high energy compaction property, i.e. it keeps as much energy as possible in few coefficients. However, one of the drawbacks of KLT is that it is non-separable transformation; thus it requires large computational resources as compared to other transform.

Discrete Fourier Transform: The DFT is linear, separable and symmetric. It also exhibits good decorrelation and energy compaction characteristics, but less so compared to the DCT. Furthermore, the DFT is a complex transformation and requires computation of both magnitude and phase information. In addition, the DFT gives rise to boundary discontinuities (Gibb's phenomena) due to its implicit periodicity [6].

Discrete Sine Transform: The DST is another option for data compression but it produces a reduced quality of reconstruction as compared to the DCT. This is due to the

fact that the DST does not yield DC coefficients and has only AC coefficients, so there is less compaction [6].

Walsh Hadamard Transform: The WHT is another method for data compression. It is fast since it requires only addition and subtraction. However, it has very low energy compaction characteristics as compared to the DCT.

Discrete Cosine Transform: The DCT represents the input data points in the form of a sum of cosine functions that are oscillating at different frequency and magnitude. In the DCT compression, almost all of the information is concentrated in a small number of low frequency coefficients. These low frequency coefficients are also known as DC components; the rest of the components are AC components. The DCT has different variations, like DCT-I, DCT-II, DCT-III and DCT-IV. Among them, only the DCT-II and the DCT-III are popular and widely used in many compression systems [34]. Therefore, we only explain the mathematical expression of these two types.

DCT-II: The most common variant of the DCT is the type-II DCT, which is often called as "DCT" by itself. The DCT-II is typically defined as a real, orthogonal (unitary), linear transformation by the formula (for $k = 0, \ldots, N-1$) [35]:

$$C_k^{II} = \sqrt{\frac{2-\delta_k}{N}} \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \tag{1.2}$$

Here, x_n = input

C_k^{II} = DCT-II transformed output

$$\delta_k = \begin{cases} 1 & ,k = 0 \\ 0 & ,otherwise \end{cases}$$

DCT-III: The type-III DCT is the inverse of type-II. It is often called as "Inverse DCT" or in short "IDCT". As both DCT-II and DCT-III follow the orthogonal property,

mathematically they are transposes of each other. We can define DCT-III (for $k = 0, \ldots, N-1$) as [35]:

$$C_k^{III} = \frac{x_0}{2} + \sqrt{\frac{2-\delta_k}{N}} \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N} n\left(k+\frac{1}{2}\right)\right] = C_k^{II\ T} \tag{1.3}$$

Here, $x_0 =$ the first input (for n = 0)

$C_k^{III} =$ DCT-III transformed output

The performance of the DCT is very close to the KLT, but requires less computational complexity [8]. In addition, the DCT has property of higher energy compaction compared to other transform [9]. Hence, the DCT has been used for image and video compression for the last decade. However, all the transform coefficients of the DCT are floating point numbers, and thus require expensive resources for implementation. This need of floating point is considered as a major drawback of the DCT from the hardware point of view. Recently a simplified version of the DCT, called Integer-DCT (Int-DCT), is introduced in which all the floating point transform coefficients of the DCT are approximated as integers [9].

Integer-DCT: All the transform coefficients of the Int-DCT are real and integer according to the name. As a result, the computational complexity is highly decreased compare to the DCT [10]. Other than this, the rest of the features are present. The Int-DCT has become very popular due to its simplified structure. Most of the modern video standards have used the Int-DCT in their compression system. The general structure of 4x4 and 8x8 inverse Int-DCT (Int-IDCT) transform matrices are shown below in Figure 1.4:

$$\begin{bmatrix} a & f & a & g \\ a & g & -a & -f \\ a & -g & -a & f \\ a & -f & a & -g \end{bmatrix} \qquad \begin{bmatrix} a & b & f & c & a & d & g & e \\ a & c & g & -e & -a & -b & -f & -d \\ a & d & -g & -b & -a & e & f & c \\ a & e & -f & -d & a & c & -g & -b \\ a & -e & -f & d & a & -c & -g & b \\ a & -d & -g & b & -a & -e & f & -c \\ a & -c & g & e & -a & b & -f & d \\ a & -b & f & -c & a & -d & g & -e \end{bmatrix}$$

(a) 4x 4 inverses transform block (b) 8x8 inverse transform block

Figure 1.4. The general matrix structure of Int-IDCT

Here, $a, b, ..., f, g$ denote seven transform coefficients which are all unique and integer in nature. All the video standards, which support Int-DCT, follow this general structure where only the coefficients ($a, b, ..., f, g$) are different. For example, the 8x8 Int-IDCT matrix of H.264/AVC video standard has the same format as Figure 1.4(b), where seven coefficients ($a, b, ..., f, g$) are respectively: 8, 12, 10, 6, 3, 8, and 4.

Discrete Wavelet Transform: The DWT represents an image as a sum of wavelet functions [58]. The basic idea of any wavelet transform is to represent an arbitrary function, $x(t)$, as a superposition of a set of such wavelets or basis functions. These basis functions or '*baby wavelets'* are obtained from a single prototype wavelet called '*mother wavelet'*, by dilations or contractions (scaling) and translations (shifts). The wavelet transform of a finite length signal, $x(t)$, is shown below [59]:

$$\psi(\tau, s) = \frac{1}{s} \int x(t) \cdot \psi^0 \left(\frac{t - \tau}{s} \right) dt \tag{1.4}$$

Here, τ = translation and s = scaling

In spatial domain, this operation is discretized. The discrete finite wavelet transform can be represented as a matrix, $\psi(c)$ and the DWT coefficients can be obtained by taking the inner product between the input signals and the wavelet matrix.

For high compression ratio, the DWT can maintain better video quality than the DCT [60]. However, the DWT requires more computational steps. The main reason is that the DCT only computes the cosine part of the fourier transform, whereas the DWT converts the entire signals from time domain to time-frequency domain [59].

1.2 The cost-shared multi-transform algorithm

In this thesis, we develop a new generalized algorithm, along with it's hardwire implementation, of a reconfigurable multi-codec transform architecture. The proposed design supports variable block size transforms which mean the user (or any other system) can configure the block size (4x4 or 8x8) during operation. The scheme is based on matrix decomposition with sparse matrices and offset computations. The sparse matrix is a type of matrix that mainly consists of zero coefficients [61]. They are advantageous for computation because of their large number of zero elements [62]. In our algorithm, these sparse matrices are derived in a way that they can be reused maximum number of times during decoding different inverse matrices. All multipliers in the design are replaced by adders and shifters. In the scheme, we first split the 8x8 transformation matrix into two small 4x4 matrices by applying permutation techniques. Then we concurrently perform separate operations on these two matrices to compute the output. One of the 4x4 sub-matrices is reused to compute the 4x4 Int-IDCT. It enables parallel operation and yields

high throughput, which eventually helps meet the coding requirement of the high resolution video.

The proposed generalized algorithm is later applied to compute the 4x4 and 8x8 Int-IDCT of AVS. Then we identify the sub-matrices of AVS and reuse them to compute the Int-IDCT of VC-1. We follow the same principle to compute the other two Int-IDCTs of H.264 and HEVC. For HEVC, we have used the transform matrices, finalized during a recent JCTVC meeting [11]. The presented scheme can be extended for larger transform units of HEVC. It can also be applied to both forward and inverse transformation; however, for this thesis we only show the implementation for the inverse process (targeted for decoders).

1.3 Previous work

In recent years, many multi-standard inverse transform architectures have been proposed for video applications. Lee's work in [12] presents a multi-standard Int-IDCT architecture based on delta coefficient matrices which can support VC-1, MPEG4 and H.264. It can process up to 21.9 fps for full HD video. Kim's work in [13] describes a design following an approach similar to [12] to unify the Int-IDCT and inverse quantization (IQ) operations for those three codecs. However, the design cannot support full HD video format. Qi's work in [14] shows an efficient integrated architecture designed for multi-standard inverse transforms of MPEG-2/4, H.264 and VC-1 using factor share (FS) and adder share (AS) strategies for saving circuit resource. The work achieves 100MHz working frequency for full HD video resolution, but does not support AVS. In another interesting design [15], the authors devise a common architecture by

sharing adders and multipliers to perform transform and quantization of H.264, MPEG-4 and VC-1.

Wahid's work [16] has developed a resource shared design using delta coefficient matrices which can compute the Int-IDCT of VC-1, JPEG, MPEG4, H.264/AVC and AVS. But due to complex data scheduling and the integration of JPEG (which is an image codec), the decoding capability is limited. The design supports both HD formats, but fails to comply with super resolution (WQXGA) and does not support 4x4 transforms. Liu [17] introduces another shared design to support Int-IDCT of VC-1, MPEG4, H.264 and AVS. Nevertheless the operating frequency of this design is low (110.8MHz) and cannot decode HD and WQXGA video. Fan's work in [18] and [19] is based on another efficient matrix decomposition algorithm to compute multiple transforms; however, the work is limited to only H.264 and VC-1. There is similar work in [20]-[22], which is also limited to these two codecs (H.264 and VC-1). One of our previous works [23] presents an efficient delta mapping sharing scheme to implement the transform unit of H.264 and HEVC. Unfortunately, this design does not support other two popular standards. In another work [24], we have proposed a hybrid 'Matrix Factorizing-Delta Mapping' scheme to combine transform unit of VC-1, JPEG, MPEG4, H.264/AVC and AVS. This design achieved very high frequency. However, it is difficult for this design to meet the power and area constrains of modern multimedia devices. In another work [25], we have resolved these drawbacks. Here, we have developed an efficient algorithm that is limited to 8x8 transforms only. Wang's work in [26] presents a reconfigurable design to compute 4x4 VC-1 and H.264 and 8x8 MPEG2, H.264, VC-1

and AVS transforms. The common shortcoming of all these designs is that none of them supports the 4x4 and 8x8 AVS (except two), nor the emerging HEVC standard.

1.4 Motivations

Recently, most of the modern video applications use one of these three video standards: VC-1 [28], H.264/AVC [27], and AVS [30]. All of them have proven their efficiency for real-time video transmission. To improve the coding efficiency further, recently a joint collaboration team on video coding (JCT-VC) is drafting a next generation video coding standard, known tentatively as High Efficiency Video Coding (HEVC) [29]. The target bit rate is half of that of H.264/AVC. Besides, several other effective techniques are proposed in the draft to reduce the complexity of the encoder, such as, improved intra-picture coding, simpler VLC coefficients [31]. As a result of these new features, experts predict that the HEVC will dominate the future multimedia market.

The inverse transform unit is a major component in a transcoder that performs a direct real-time digital-to-digital data conversion from one encoding to another [56]. As a result, in recent years, there is a growing interest to develop multi-standard inverse transform architectures for advanced multimedia transcoding applications. However, most of them do not support AVS (also known as AVS-P2 or Jizhun Profile) and AVS-P7 (a subset of AVS that is targeted for mobile applications [32]). AVS is the video codec developed by Chinese government that became the core technology of China Mobile Multimedia Broadcasting (CMMB) [53]. Moreover, none of the existing work supports HEVC. Although presently in draft stage, considering the future prospective of

the HEVC, it is important to start exploring possible hardware implementation of the transform unit discussed in the draft.

Considering all these facts, we have been motivated to combine these four video standards (AVS, H.264, VC-1, and HEVC) under a unified shared architecture that can perform variable block size (4x4 and 8x8) transformation for all.

1.5 Thesis objective

The objective of this thesis is to develop a unified architecture on a single chip that perform both 4x4 and 8x8 Int-DCT based transform for the four modern video standards (AVS, H.264, VC-1, and HEVC). Our focus is on the efficient implementation of multiple transform units; the implementation of the full encoder (or decoder) is not explored. However, integrating multiple transform units of different standards into a single chip increases the area and decreases the frequency, which has negative impact on the overall performance. Therefore, the ultimate goal is to design and implement a low cost multiple transform unit that meets the real-time performance requirements. To accomplish this goal, the thesis work is focused on these points:

I. Investigate different transform coding techniques of video compression system. The intension is to find the most efficient transform technique which can be applied to all the four video standards (AVS, H.264, VC-1, and HEVC).

II. Develop an efficient sharing algorithm for 4x4 block size. Then extend the algorithm for 8x8 block size.

III. Simulate the design to verify the functionality. Then synthesize the design to assess the hardware cost.

IV. Demonstrate the performance advantages of the proposed architecture in terms of cost and decoding performance.

As a part of this thesis, the developed algorithm is modeled in Verilog HDL, then implemented on a FPGA (Virtex 4 LX60) and synthesized on $0.18\mu m$ CMOS technology.

1.6 Thesis organization

This thesis is organized into six chapters. In Chapter 2, we briefly present the overview of four modern video codecs: AVS, H.264, VC-1, and HEVC. The development of proposed sharing algorithm is presented in Chapter 3. Chapter 4 presents the architectural details, hardware mapping and sharing mechanism. The simulation and synthesis results of FPGA and $0.18\mu m$ CMOS technology along with their performance analysis are summarized in Chapter 5. Chapter 6 concludes the thesis by summarizing the accomplishments of the research work and giving recommendation for future exploration.

CHAPTER 2

OVERVIEW OF MODERN VIDEO CODECS

2.1 Introduction

This chapter presents an overview of four video codecs- H.264/AVC, VC-1, AVS, and HEVC (under development). The structure of all these video codecs are very similar. At first, the major components of H.264/AVC codec and their functionalities are briefly described to provide a general idea about how the modern video codes function. Next short descriptions of the transform units are presented. Finally, a comparison of the transform block sizes and their implementation cost are presented.

2.2 Overview of H.264/AVC codec

H.264/AVC is considered the current state-of-the-art video coding standards. The official name of the H.264/AVC is Advanced Video Coding (AVC) of MPEG-4 part 10 in ISO/IEC and H.264 in ITU-T [36].

This video coding standard has the same basic functional elements as previous standards (MPEG-1, MPEG-2, MPEG-4 part 2, H.261, and H.263) [37], i.e., transform for reduction of spatial correlation, quantization for bit rate control, motion compensated prediction for reduction of temporal correlation, and entropy encoding for reduction of statistical correlation. However, to fulfill better coding performance, the important changes in H.264 occur in the details of each functional element by including intra-picture prediction, a new 4x4 integer transform, multiple reference pictures, variable block sizes and a quarter pixel precision for motion compensation, a deblocking filter,

and improved entropy coding. The block diagram of H.264 encoder and decoder are shown below in Figure 2.1 and Figure 2.2:

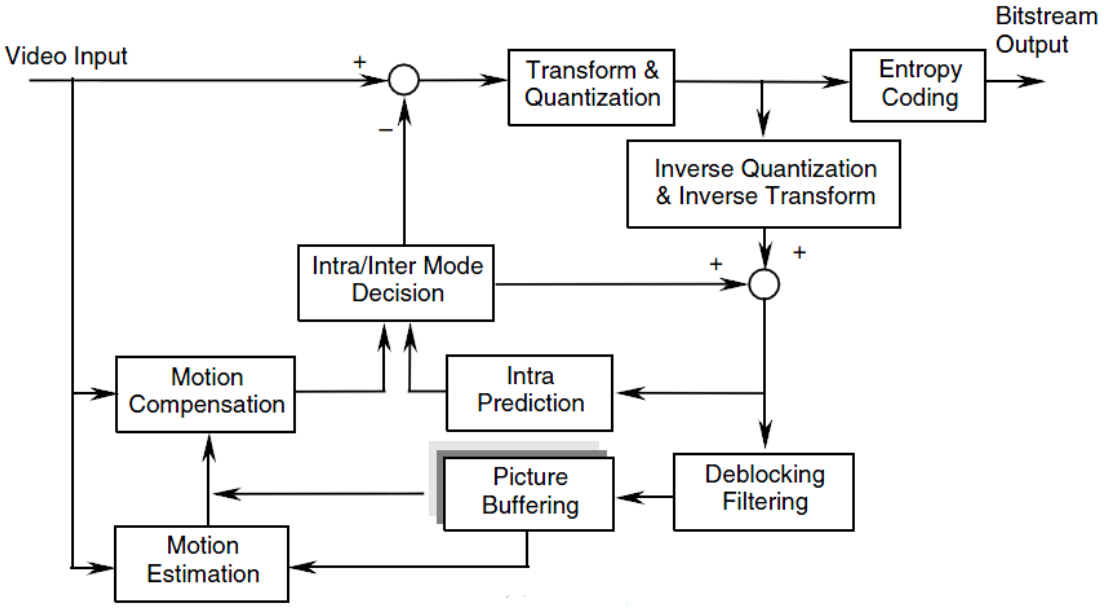

Figure 2.1. Block diagram of H.264 encoder [37]

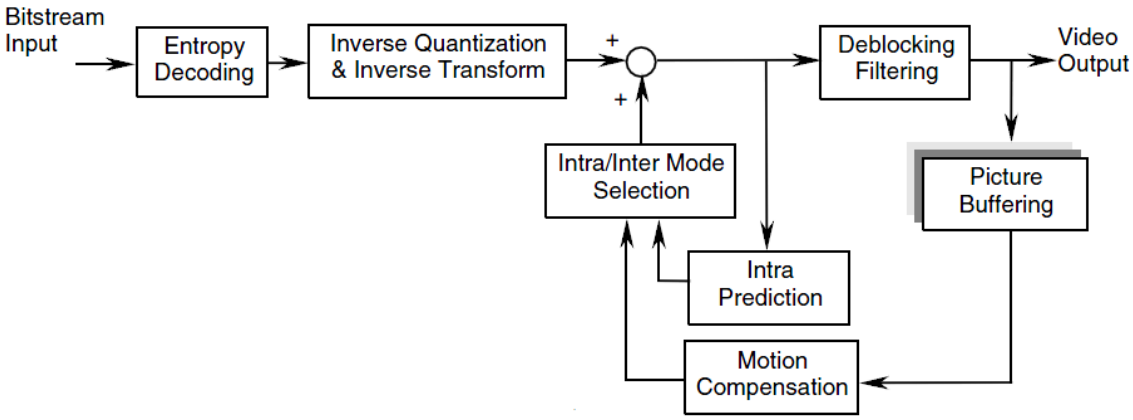

Figure 2.2. Block diagram of H.264 decoder [37]

The H.264/AVC encoder can be understood from its typical structural diagram as illustrated in Figure 2.1. The encoding operation starts by splitting the first frame of a sequence or random access point into macroblocks (MB). This frame is usually

intracoded with no use of reference frames. The samples in a block are predicted from previously encoded neighboring blocks. The encoding process selects the best neighboring block and determines how the samples from these blocks are combined for inter prediction. The decoder is notified of this selection process.

The decoder performs an inverse of the operations done by the encoder. It inverts the entropy coding process and then using the motion data and the type of prediction performs the prediction process. Inverse scaling and transforming of the residual is also done and the deblocking filter is applied to the result to get the video output. A block diagram of H.264/AVC decoder is shown in Figure 2.2.

2.3 Transform unit of H.264/AVC

H.264/AVC standard is based on the use of a block-based transform for spatial redundancy removal. It uses an adaptive transform block size, 4x4 and 8x8 (High Profiles only), whereas previous video coding standards used the 8x8 DCT. The smaller block size leads to a significant reduction in ringing artifacts. Also, the 4x4 transform has the additional benefit of removing the need for multiplications. For improved compression efficiency, H.264 also employs a hierarchical transform structure, in which the DC coefficient of neighboring 4x4 transforms for the luma signals are grouped into 4x4 blocks. Figure 2.4 shows the 4x4 and 8x8 inverse transform matrices of H.264/AVC:

$$\begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

(a) 4x4 Int-IDCT matrix

$$\begin{bmatrix} 8 & 12 & 8 & 10 & 8 & 6 & 4 & 3 \\ 8 & 10 & 4 & -3 & -8 & -12 & -8 & -6 \\ 8 & 6 & -4 & -12 & -8 & 3 & 8 & 10 \\ 8 & 3 & -8 & -6 & 8 & 10 & -4 & -12 \\ 8 & -3 & -8 & 6 & 8 & -10 & -4 & 12 \\ 8 & -6 & -4 & 12 & -8 & -3 & 8 & -10 \\ 8 & -10 & 4 & 3 & -8 & 12 & -8 & 6 \\ 8 & -12 & 8 & -10 & 8 & -6 & 4 & -3 \end{bmatrix}$$

(b) 8x8 Int-IDCT matrix

Figure 2.3. Inverse transform matrices of H.264/AVC codec

There are much work available in literature on efficient implementation of the standalone 4x4 and 8x8 transform unit of H.264/AVC. Among them, Malav [38] proposed a 4x4 Int-DCT transform architecture of H.264 which is multiplier less and suitable for low-end processors. It required only 8 addition and 2 shift operations to complete the operation. Gordon [39] proposed another simplified design to implement 8x8 Int-DCT of H.264. The computational complexity of this design is 32 additions and 10 shift operations. In [16], the authors have computed the standalone 8x8 Int-DCT of H.264 by performing only 24 addition and 11 shift operations.

2.4 Transform unit of VC-1

VC-1, officially known as Society of Motion Picture and Television Engineers (SMPTE) 421M, was initially developed as a proprietary video format by Microsoft, was released as a SMPTE video codec standard on April 3, 2006 [40]. Hence this codec is also known as Microsoft Windows Media Video 9 (WMV-9).

Traditionally, 8x8 transforms have been used for most of the previous image and video coding [41]-[43]. The 8x8 size has the advantages of being dyadic, large enough to capture trends and periodicities while being small enough to minimize spreading effects due to local transients over the transform area. Trends and textures, especially periodic textures, are better preserved with the transforms having a larger support. On the other hand, it is known that smaller transforms are better in areas with discontinuities because they produce fewer ringing artifacts [44]-[46]. Hence VC-1 has defined both 4x4 and 8x8 size transform matrices to provide the flexibility to choose the transform that is best suited for the underlying data. Figure 2.4 shows the 4x4 and 8x8 inverse transform matrices of VC-1:

$$\begin{bmatrix} 17 & 22 & 17 & 10 \\ 17 & 10 & -17 & -22 \\ 17 & -10 & -17 & 22 \\ 17 & -22 & 17 & -10 \end{bmatrix}$$

(a) 4x4 Int-IDCT matrix

$$\begin{bmatrix} 12 & 16 & 16 & 15 & 12 & 9 & 6 & 4 \\ 12 & 15 & 6 & -4 & -12 & -16 & -16 & -9 \\ 12 & 9 & -6 & -16 & -12 & 4 & 16 & 15 \\ 12 & 4 & -16 & -9 & 12 & 15 & -6 & -16 \\ 12 & -4 & -16 & 9 & 12 & -15 & -6 & 16 \\ 12 & -9 & -6 & 16 & -12 & -4 & 16 & -15 \\ 12 & -15 & 6 & 4 & -12 & 16 & -16 & 9 \\ 12 & -16 & 16 & -15 & 12 & -9 & 6 & -4 \end{bmatrix}$$

(b) 8x8 Int-IDCT matrix

Figure 2.4. Inverse transform matrices of VC-1 codec

Srinivasan [47] has implemented a standalone 8x8 transform unit with a sequence of butterfly operations and multiplies. The hardware cost of that design is 38 additions and 18 shift operations. In that work he also computed the implement cost of 4x4 VC-1 transform unit as 16 additions and 12 shifts. Fan proposed another fast algorithm [19] based on a novel ‘matrix decomposition’ algorithm to compute a standalone 8x8 VC-1

transform unit with 36 addition operations. In [16], we have computed the standalone 8x8 Int-DCT architecture of VC-1 with the minimum cost. We performed 24 addition and 12 shift operations for this Int-DCT operation.

2.5 Transform unit of AVS

The Advance Video coding Standard (AVS) is based on the classic hybrid differential pulse code modulation - discrete cosine transform (DPCM-DCT) coder, which was first introduced by Jain and Jain in 1979 [6]. AVS has 10 different parts, among them AVS-Part2 (AVS-P2) and AVS-Part7 (AVS-P7) are dedicated for video compression [30].

As larger transform size is more efficient for high resolution coding, AVS-P2 has 8×8 transform matrix. Figure 2.5 shows the inverse transform matrix of AVS-P2:

$$\begin{bmatrix} 8 & 10 & 10 & 9 & 8 & 6 & 4 & 2 \\ 8 & 9 & 4 & -2 & -8 & -10 & -10 & -6 \\ 8 & 6 & -4 & -10 & -8 & 2 & 10 & 9 \\ 8 & 2 & -10 & -6 & 8 & 9 & -4 & -10 \\ 8 & -2 & -10 & 6 & 8 & -9 & -4 & 10 \\ 8 & -6 & -4 & 10 & -8 & -2 & 10 & -9 \\ 8 & -9 & 4 & 2 & -8 & 10 & -10 & 6 \\ 8 & -10 & 10 & -9 & 8 & -6 & 4 & -2 \end{bmatrix}$$

Figure 2.5. Inverse transform matrices of AVS-P2 codec

The transform of AVS-P7 has the similar feature as that in WMV9, called Pre-scaled Integer Transform (PIT) [49] [50]. As the basis of the transform coefficients is very close, the transform normalization can be accounted entirely on the encoder side. The transform in AVS-P7 has the same feature as that in AVS-P2, but a 4×4 transform is used. The inverse transform matrix of AVS-P7 is shown below:

$$\begin{bmatrix} 2 & 3 & 2 & 1 \\ 2 & 1 & -2 & -3 \\ 2 & -1 & -2 & 3 \\ 2 & -3 & 2 & -1 \end{bmatrix}$$

Figure 2.6. Inverse transform matrices of AVS-P7 codec

The implementation cost of standalone AVS-P2 and AVS-P7 transform unit can be estimated from our work [25]. At this design all the multipliers are replaced by equivalent addition and shift operations to make the hardware architecture simpler. There we implement the Int-DCT transform unit of AVS-P2 and AVS-P7 by using 24 and 9 adders respectively.

2.6 Transform unit of HEVC

The High Efficiency Video Coding (HEVC) standard is currently under development by the JCT-VC group (jointly created by ISO/IEC MPEG and ITU-T VCEG) and it is planned to be ratified as a standard by January 2013 [51].

A larger transform can bring high performance improvements in terms of energy compaction and reduced quantization error for large homogeneous areas. HD sequences tend to have more spatial correlation, this means in larger parts of the image. Thus, HEVC introduces three additional transform sizes besides those already supported by H.264/AVC (4×4 and 8×8): 16×16, and 32×32. Figure 2.7 and Figure 2.8 show different block size inverse transform matrices of HEVC which are finalized by recent meeting of JCT-VC [52]. As our goal is to design a sheared mulit-codec transform unit rather than the standalone implementation of HEVC, in this thesis we will only perform the 4x4 and 8x8 Int-IDCT of this codec.

$$\begin{bmatrix} 64 & 83 & 64 & 36 \\ 64 & 36 & -64 & -83 \\ 64 & -36 & -64 & 83 \\ 64 & -83 & 64 & -36 \end{bmatrix}$$

(a) 4x4 Int-IDCT matrix

$$\begin{bmatrix} 64 & 89 & 83 & 75 & 64 & 50 & 36 & 18 \\ 64 & 75 & 36 & -18 & -64 & -89 & -83 & -50 \\ 64 & 50 & -36 & -89 & -64 & 18 & 83 & 75 \\ 64 & 18 & -83 & -50 & 64 & 75 & -36 & -89 \\ 64 & -18 & -83 & 50 & 64 & -75 & -36 & 89 \\ 64 & -50 & -36 & 89 & -64 & -18 & 83 & -75 \\ 64 & -75 & 36 & 18 & -64 & 89 & -83 & 50 \\ 64 & -89 & 83 & -75 & 64 & -50 & 36 & -18 \end{bmatrix}$$

(b) 8x8 Int-IDCT matrix

$$\begin{bmatrix} 64 & 90 & 89 & 87 & 83 & 80 & 75 & 70 & 64 & 57 & 50 & 43 & 36 & 25 & 18 & 9 \\ 64 & 87 & 75 & 57 & 36 & 9 & -18 & -43 & -64 & -80 & -89 & -90 & -83 & -70 & -50 & -25 \\ 64 & 80 & 50 & 9 & -36 & -70 & -64 & -87 & -64 & -25 & 18 & 57 & 83 & 90 & 75 & 43 \\ 64 & 70 & 18 & -43 & -83 & -87 & -50 & 9 & 64 & 90 & 75 & 25 & -36 & -80 & -83 & -57 \\ 64 & 57 & -18 & -80 & -83 & -25 & 50 & 90 & 64 & -9 & -75 & -87 & -36 & 43 & 83 & 70 \\ 64 & 43 & -50 & -90 & -36 & 57 & 89 & 25 & -64 & -87 & -18 & 70 & 83 & 9 & -75 & -80 \\ 64 & 25 & -75 & -70 & 36 & 90 & 18 & -80 & -64 & 43 & 89 & 9 & -83 & -57 & 50 & 87 \\ 64 & 9 & -89 & -25 & 83 & 43 & -75 & -57 & 64 & 70 & -50 & -80 & 36 & 87 & -18 & -90 \\ 64 & -9 & -89 & 25 & 83 & -43 & -75 & 57 & 64 & -70 & -50 & 80 & 36 & -87 & -18 & 90 \\ 64 & -25 & -75 & 70 & 36 & -90 & 18 & 80 & -64 & -43 & 89 & -9 & -83 & 57 & 50 & -87 \\ 64 & -43 & -50 & 90 & -36 & -57 & 89 & -25 & -64 & 87 & -18 & -70 & 83 & -9 & -75 & 80 \\ 64 & -57 & -18 & 80 & -83 & 25 & 50 & -90 & 64 & 9 & -75 & 87 & -36 & -43 & 89 & -70 \\ 64 & -70 & 18 & 43 & -83 & 87 & -50 & -9 & 64 & -90 & 75 & -25 & -36 & 80 & -89 & 57 \\ 64 & -80 & 50 & -9 & -36 & 70 & -89 & 87 & -64 & 25 & 18 & -57 & 83 & -90 & 75 & -43 \\ 64 & -87 & 75 & -57 & 36 & -9 & -18 & 43 & -64 & 80 & -89 & 90 & -83 & 70 & -50 & 25 \\ 64 & -90 & 89 & -87 & 83 & -80 & 75 & -70 & 64 & -57 & 50 & -43 & 36 & -25 & 18 & -9 \end{bmatrix}$$

(c) 16x16 Int-IDCT matrix

Figure 2.7. Inverse transform matrices of HEVC codec (8x8 and 16x16)

$$
\begin{bmatrix}
64 & 90 & 90 & 90 & 89 & 88 & 87 & 85 & 83 & 82 & 80 & 78 & 75 & 73 & 70 & 67 & 64 & 61 & 57 & 54 & 50 & 46 & 43 & 38 & 36 & 31 & 25 & 22 & 18 & 13 & 9 & 4 \\
64 & 90 & 87 & 82 & 75 & 67 & 57 & 46 & 36 & 22 & 9 & -4 & -18 & -31 & -43 & -54 & -64 & -73 & -80 & -85 & -89 & -90 & -90 & -88 & -83 & -78 & -70 & -61 & -50 & -38 & -25 & -13 \\
64 & 88 & 80 & 67 & 50 & 31 & 9 & -13 & -36 & -54 & -70 & -82 & -89 & -90 & -87 & -78 & -64 & -46 & -25 & -4 & 18 & 38 & 57 & 73 & 83 & 90 & 90 & 85 & 75 & 61 & 43 & 22 \\
64 & 85 & 70 & 46 & 18 & -13 & -43 & -67 & -83 & -90 & -87 & -73 & -50 & -22 & 9 & 38 & 64 & 82 & 90 & 88 & 75 & 54 & 25 & -4 & -36 & -61 & -80 & -90 & -89 & -78 & -57 & -31 \\
64 & 82 & 57 & 22 & -18 & -54 & -80 & -90 & -83 & -61 & -25 & 13 & 50 & 78 & 90 & 85 & 64 & 31 & -9 & -46 & -75 & -90 & -87 & -67 & -36 & 4 & 43 & 73 & 89 & 88 & 70 & 38 \\
64 & 78 & 43 & -4 & -50 & -82 & -90 & -73 & -36 & 13 & 57 & 85 & 89 & 67 & 25 & -22 & -64 & -88 & -87 & -61 & -18 & 31 & 70 & 90 & 83 & 54 & 9 & -38 & -75 & -90 & -80 & -46 \\
64 & 73 & 25 & -31 & -75 & -90 & -70 & -22 & 36 & 78 & 90 & 67 & 18 & -38 & -80 & -90 & -64 & -13 & 43 & 82 & 89 & 61 & 9 & -46 & -83 & -66 & -57 & -4 & 50 & 85 & 87 & 54 \\
64 & 67 & 9 & -54 & -89 & -78 & -25 & 38 & 83 & 85 & 43 & -22 & -75 & -90 & -57 & 4 & 64 & 90 & 70 & 13 & -50 & -88 & -80 & -31 & 36 & 82 & 87 & 46 & -18 & -73 & -90 & -61 \\
64 & 61 & -9 & -73 & -89 & -46 & 25 & 82 & 83 & 31 & -43 & -88 & -75 & -13 & 57 & 90 & 64 & -4 & -70 & -90 & -50 & 22 & 80 & 85 & 36 & -38 & -87 & -73 & -18 & 54 & 90 & 67 \\
64 & 54 & -25 & -85 & -75 & -4 & 70 & 88 & 36 & -46 & -90 & -61 & 18 & 82 & 80 & 13 & -64 & -90 & -43 & 38 & 89 & 67 & -9 & -78 & -83 & -22 & 57 & 90 & 50 & -31 & -87 & -73 \\
64 & 46 & -43 & -90 & -50 & 38 & 90 & 54 & -36 & -90 & -57 & 31 & 89 & 61 & -25 & -88 & -64 & 22 & 87 & 67 & -18 & -85 & -70 & 13 & 83 & 73 & -9 & -86 & -75 & 4 & 80 & 78 \\
64 & 38 & -57 & -88 & -18 & 73 & 80 & -4 & -83 & -67 & 25 & 90 & 50 & -46 & -90 & -31 & 64 & 85 & 9 & -78 & -75 & 13 & 87 & 61 & -36 & -90 & -43 & 54 & 89 & 22 & -70 & -82 \\
64 & 31 & -70 & -78 & 18 & 90 & 43 & -61 & -83 & 4 & 87 & 54 & -50 & -88 & -9 & 82 & 64 & -38 & -90 & -22 & 75 & 73 & -25 & -90 & -36 & 67 & 80 & -13 & -89 & -46 & 57 & 85 \\
64 & 22 & -80 & -61 & 50 & 85 & -9 & -90 & -36 & 73 & 70 & -38 & -89 & -4 & 87 & 46 & -64 & -78 & 25 & 90 & 18 & -82 & -57 & 54 & 83 & -13 & -90 & -31 & 75 & 67 & -43 & -88 \\
64 & 13 & -87 & -38 & 75 & 61 & -57 & -78 & 36 & 88 & -9 & -90 & -18 & 85 & 43 & -73 & -64 & 54 & 80 & -31 & -89 & 4 & 90 & 22 & -83 & -46 & 70 & 67 & -50 & -82 & 25 & 90 \\
64 & 4 & -90 & -13 & 89 & 22 & -87 & -31 & 83 & 38 & -80 & -46 & 75 & 54 & -70 & -61 & 64 & 67 & -57 & -73 & 50 & 78 & -43 & -82 & 36 & 85 & -25 & -88 & 18 & 90 & -9 & -90 \\
64 & -4 & -90 & 13 & 89 & -22 & -87 & 31 & 83 & -38 & -80 & 46 & 75 & -54 & -70 & 61 & 64 & -67 & -57 & 73 & 50 & -78 & -43 & 82 & 36 & -85 & -25 & 88 & 18 & -90 & -9 & 90 \\
64 & -13 & -87 & 38 & 75 & -61 & -57 & 78 & 36 & -88 & -9 & 90 & -18 & -85 & 43 & 73 & -64 & -54 & 80 & 31 & -89 & -4 & 90 & -22 & -83 & 46 & 70 & -67 & -50 & 82 & 25 & -90 \\
64 & -2 & -80 & 61 & 50 & -85 & -9 & 90 & -36 & -73 & 70 & 38 & -89 & 4 & 87 & -46 & -64 & 78 & 25 & -90 & 18 & 82 & -57 & -54 & 83 & 13 & -90 & 31 & 75 & -67 & -43 & 88 \\
64 & -31 & -70 & 78 & 18 & -90 & 43 & 61 & -83 & -4 & 87 & -54 & -50 & 88 & -9 & -82 & 64 & 38 & -90 & 22 & 75 & -73 & -25 & 90 & -36 & -67 & 80 & 13 & -89 & 46 & 57 & -85 \\
64 & -38 & -57 & 88 & -18 & -73 & 80 & 4 & -83 & 67 & 25 & -90 & 50 & 46 & -90 & 31 & 64 & -85 & 9 & 78 & -75 & -13 & 87 & -61 & -36 & 90 & -43 & -54 & 89 & -22 & -70 & 82 \\
64 & -46 & -43 & 90 & -50 & -38 & 90 & -54 & -36 & 90 & -57 & -31 & 89 & -61 & -25 & 88 & -64 & -22 & 87 & -67 & -18 & 85 & -70 & -13 & 83 & -73 & -9 & 82 & -75 & -4 & 80 & -78 \\
64 & -54 & -25 & 85 & -75 & 4 & 70 & -88 & 36 & 46 & -90 & 61 & 18 & -82 & 80 & -13 & -64 & 30 & -43 & -38 & 89 & -67 & -9 & 78 & -83 & 22 & 57 & -90 & 50 & 31 & -87 & 73 \\
64 & -61 & -9 & 73 & -89 & 46 & 25 & -82 & 83 & -31 & -43 & 88 & -75 & 13 & 57 & -90 & 64 & 4 & -70 & 90 & -50 & -22 & 80 & -85 & 36 & 38 & -87 & 78 & -18 & -54 & 90 & -67 \\
64 & -67 & 9 & 54 & -89 & 78 & -25 & -38 & 83 & -85 & 43 & 22 & -75 & 90 & -57 & -4 & 64 & -90 & 70 & -13 & -50 & 88 & -80 & 31 & 36 & -82 & 87 & -46 & -18 & 73 & -90 & 61 \\
64 & -73 & 25 & 31 & -75 & 90 & -70 & 22 & 36 & -78 & 90 & -67 & 18 & 38 & -80 & 90 & -64 & 13 & 43 & -82 & 89 & -61 & 9 & 46 & -83 & 88 & -57 & 4 & 50 & -85 & 87 & -54 \\
64 & -78 & 43 & 4 & -50 & 82 & -90 & 73 & -36 & -13 & 57 & -85 & 89 & -67 & 25 & 22 & -64 & 88 & -87 & 61 & -18 & -31 & 70 & -90 & 83 & -54 & 9 & 38 & -75 & 90 & -80 & 46 \\
64 & -82 & 57 & -22 & -18 & 54 & -80 & 90 & -83 & 61 & -25 & -13 & 50 & -78 & 90 & -85 & 64 & -31 & -9 & 46 & -75 & 90 & -87 & 67 & -36 & -4 & 43 & -73 & 89 & -88 & 70 & -38 \\
64 & -85 & 70 & -46 & 18 & 13 & -43 & 67 & -83 & 90 & -87 & 73 & -50 & 22 & 9 & -38 & 64 & -82 & -90 & -88 & 75 & -54 & 25 & 4 & -36 & 61 & -80 & 90 & -89 & 73 & -57 & 31 \\
64 & -88 & 80 & -67 & 50 & -31 & 9 & 13 & -36 & 54 & -70 & 82 & -89 & 90 & -87 & 78 & -64 & 46 & -25 & 4 & 18 & -38 & 57 & -73 & 83 & -90 & 90 & -85 & 75 & -61 & 43 & -22 \\
64 & -90 & 87 & -82 & 75 & -67 & 57 & -46 & 36 & -22 & 9 & 4 & -18 & 31 & -43 & 54 & -64 & 73 & -80 & 85 & -86 & 90 & -90 & 88 & -83 & 78 & -70 & 61 & -50 & 38 & -25 & 13 \\
64 & -92 & 90 & -90 & 89 & -88 & 87 & -85 & 83 & -82 & 80 & -78 & 75 & -73 & 70 & -67 & 64 & -61 & 57 & -54 & 50 & -46 & 43 & -38 & 36 & -31 & 25 & -22 & 18 & -13 & 9 & -4
\end{bmatrix}
$$

(d) 32x32 Int-IDCT matrix

Figure 2.8. Inverse transform matrix of HEVC codec (32x32)

The hardware cost of 8x8 standalone HEVC transform unit can be calculated from our work [23]. That design cost 33 addition and 17 shift operations to implement a 8x8 Int-DCT architecture of HEVC. Similarly, we can get the computational cost of the 4x4 block size transform unit of HEVC from another work [25]. According to this work, we require 12 adders to implement the 4x4 Int-DCT unit of HEVC. As HEVC is very new and not yet standardized, till now there is no existing work on implementation of other block size transform unit of this codec.

2.7 Comparison of transform units

The transform block size and their computational cost (in term of adders) for all the above discussed video codecs are summarized in Table 2-1:

Table 2-1: Comparison of transform unit

	H.264 /AVC	**VC-1**	**AVS**	**HEVC (not standardized)**
Supported Transform Size [54]	Main: 4x4 High: 4x4 & 8x8	4x4, 8x8, 8x4 & 4x8	AVS-P2: 8x8 AVS-P7: 4x4	4x4, 8x8, 16x16 & 32x32
Adder count [16]	For 4x4: 8 For 8x8: 24	For 4x4: 12 For 8x8: 24	For 4x4: 9 For 8x8: 24	For 4x4: 12 For 8x8: 32

It is visible from the first row of this table that all the four standards support the Int-DCT because of its simple implementation. We can also observe that 4x4 and 8x8 are the common transform block size among them. Only the new codec HEVC supports larger size blocks (16x16 and 32x32).

The second row of Table 2-1 shows the number of performed addition operations to implement 4x4 and 8x8 transform unit of different standards. As the number of arithmetic unit, like addition, is proportionate to the hardware resource, from this row we can compare their computational cost. It is observed that implementation of HEVC requires the highest number of adders; the reason is its transform coefficients ($a, b, ..., f, g$) are larger than those of other three codecs.

CHAPTER 3

PROPOSED SHARING ALGORITHM

3.1 Introduction

In this chapter, we present the detail of our proposed hardware sharing algorithm to design a unified architecture that can perform both 4x4 and 8x8 Int-IDCT of the four video codec: AVS, VC-1, H.264 and HEVC. First we develop the sharing algorithm for the 4x4 transform unit. Then we extend it for the 8x8 transform unit.

3.2 Proposed algorithm for 4x4 Int-DCT

In a video compression system, the 4x4 transform employs a 4x4 type-II DCT. The general expression of a 4x4 1D Integer Inverse DCT (IDCT) coefficient matrix is expressed below in Eqn. (3.1):

$$I_{4\times4} = \begin{bmatrix} a & f & a & g \\ a & g & -a & -f \\ a & -g & -a & f \\ a & -f & a & -g \end{bmatrix} \tag{3.1}$$

Here, a, f, g denote three transform coefficients. In this thesis, we have denoted the 4x4 IDCT transform matrices for AVS-P7, VC-1, H.264 and HEVC by $A_{4\times4}$, $V_{4\times4}$, $H_{4\times4}$, and $HV_{4\times4}$ respectively. These coefficients (a, f, g) for each of the transforms are different, but integer in nature (as shown in Table 3-1). Since, the forward DCT (FDCT)

uses the same basis coefficients and is the transpose of the inverse DCT matrix, the proposed IDCT scheme is easily applicable to FDCT without any added cost or complexity.

Table 3-1: Matrix coefficients of 4x4 Int-DCT

	AVS-P7	VC-1	H.264	HEVC
a	2	17	1	64
f	3	22	2	83
g	1	10	1	36

3.2.1 Matrix decomposition for 4x4 AVS-P7

First of all, we construct $A_{4\times4}$ according to Eqn. (3.1) and Table 3-1, and then decompose it with the help of two simple sparse matrices A_1 and A_2 as shown below:

$$A_{4\times4} = \begin{bmatrix} 2 & 3 & 2 & 1 \\ 2 & 1 & -2 & -3 \\ 2 & -1 & -2 & 3 \\ 2 & -3 & 2 & -1 \end{bmatrix} = A_1 \cdot 2A_2 \tag{3.2}$$

$$\text{Here, } A_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix} \text{ and } A_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & \frac{3}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & -\frac{3}{2} \end{bmatrix}$$

The computational cost of A_1 is only 4 additions. For A_2, we implement $(3/2)\cdot x$ as $(1+1/2).x$ – that is right shift *x* (where *x* is an arbitrary data input) by one bit and add with *x*. So, the total cost is 6 addition and 4 shift operations. We require another 4 shifters to compute $2A_2$. Thus in Eqn. (3.2), the total computational cost is 10 addition and 8 shift operations.

3.2.2 Matrix decomposition for 4x4 VC-1

For 4x4 VC-1, we decompose $V_{4\times4}$ in similar way as shown below:

$$V_{4\times4} = \begin{bmatrix} 17 & 22 & 17 & 10 \\ 17 & 10 & -17 & -22 \\ 17 & -10 & -17 & 22 \\ 17 & -22 & 17 & -10 \end{bmatrix} = 8 \cdot A_1 \cdot V_2 \tag{3.3}$$

Where , $V_2 = \begin{bmatrix} \frac{17}{8} & 0 & \frac{17}{8} & 0 \\ \frac{17}{8} & 0 & -\frac{17}{8} & 0 \\ 0 & \frac{11}{4} & 0 & \frac{5}{4} \\ 0 & \frac{5}{4} & 0 & -\frac{11}{4} \end{bmatrix} = 2A_2 - V_3$ and $V_3 = \frac{1}{4} \cdot \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

So, we can re-express Eqn. (3.3) as:

$$V_{4\times4} = 8 \cdot A_1 \cdot \left(2A_2 - V_3\right) \tag{3.4}$$

Now it can be seen that the components of 4x4 AVS-P7 matrix (A_1 and $2A_2$ from Eqn. (3.2)) can be reused in Eqn. (3.4). This matrix decomposition enables hardware sharing and results in significant saving in implementation resources. Here to implement V_3, we require 4 addition and 8 shift operations. Hence, the total cost in Eqn. (3.4) is 8 additions and 12 shift operations.

3.2.3 Matrix decomposition for 4x4 H.264/AVC

We now decompose the 4x4 H.264/AVC matrix $H_{4\times4}$ as below:

$$H_{4\times4} = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} = A_1 \cdot A_{2h-4P} \tag{3.5}$$

Where, $A_{2h-4P} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 1 & 0 & -2 \end{bmatrix}$

In Eqn. (3.5), A_1 is reused from Eqn. (3.2). To share A_{2h-4p} from the architecture of A_2, we simply add four multiplexer units to replace $(3/2)\cdot x$ and $(1/2)\cdot x$ coefficients by $2\cdot x$ and x respectively. So we require only 2 shift (and no addition) operations to compute $H_{4\times4}$.

3.2.4 Matrix decomposition for 4x4 HEVC

Based on similar principle, as described above, we simplify $HV_{4\times4}$ as expressed below:

$$HV_{4\times4} = \begin{bmatrix} 64 & 83 & 64 & 36 \\ 64 & 36 & -64 & -83 \\ 64 & -36 & -64 & 83 \\ 64 & -83 & 64 & -36 \end{bmatrix} = 4\cdot A_1 . (16A_2 + HV_1) \tag{3.6}$$

Where $HV_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{3}{4} & 0 & 1 \\ 0 & 1 & 0 & -\frac{3}{4} \end{bmatrix}$

In Eqn. (3.6), the computational cost of the only new matrix, HV_1 is 4 addition and 2 shift operations. Here the coefficient (3/4)..x is factorized as (x - x/4). So the cost to implement $HV_{4\times4}$ in Eqn. (3.6) is 8 addition and 10 shift operations.

Therefore, the proposed cost-shared algorithm, described from section 3.2.1 to 3.2.4, requires total 26 addition and 32 shift operations to implement the 4x4 int-IDCTs of AVS-P7, VC-1, H.264 and HEVC.

3.3 Proposed algorithm for 8x8 Int-DCT

In a multiple standard video decoder, both 4x4 and 8x8 transform operations are required for transcoding. As a result, in this section we extend the decomposition scheme to compute 8x8 Int-IDCT coefficients. Here, we firstly propose a generalized "decompose and share" algorithm, which is later applied to all four codecs. The objective is to decompose the 8x8 transformation matrices in such a way that sharing is maximized (by reuse already developed 4x4 transform units from Section 3.2). The 8x8 Int-IDCT matrix is expressed in general form as shown below in (3.7):

$$I_{8\times8} = \begin{bmatrix} a & b & f & c & a & d & g & e \\ a & c & g & -e & -a & -b & -f & -d \\ a & d & -g & -b & -a & e & f & c \\ a & e & -f & -d & a & c & -g & -b \\ a & -e & -f & d & a & -c & -g & b \\ a & -d & -g & b & -a & -e & f & -c \\ a & -c & g & e & -a & b & -f & d \\ a & -b & f & -c & a & -d & g & -e \end{bmatrix} \tag{3.7}$$

Here, $a, b, c..., g$ denote seven transform coefficients which are different for all the standards. Table 3-2 illustrates detail of these coefficients for the four video standards:

Table 3-2: Matrix coefficients of 8x8 Int-DCT

	AVS-P2	VC-1	H.264	HEVC
a	8	12	8	64
b	10	16	12	89
c	9	15	10	75
d	6	9	6	50
e	2	4	3	18
f	10	16	8	83
g	4	6	4	36

For consistency, we denote the 8×8 matrices for AVS-P2, VC-1, H.264/AVC and HEVC by $A_{8\times8}$, $V_{8\times8}$, $H_{8\times8}$, and $HV_{8\times8}$ respectively in this chapter.

3.3.1 Development of a generalized "*decompose and share*" algorithm

In this section, we derive a generalized matrix decomposition scheme by utilizing the symmetric structure of the matrices and factoring the 8x8 matrix into two 4x4 sub-matrices as shown below:

$$I_{8\times8} = P_0 \cdot I_0 \tag{3.8}$$

Where $I_0 = \begin{bmatrix} a & 0 & f & 0 & a & 0 & g & 0 \\ a & 0 & g & 0 & -a & 0 & -f & 0 \\ a & 0 & -g & 0 & -a & 0 & f & 0 \\ a & 0 & -f & 0 & a & 0 & -g & 0 \\ 0 & -e & 0 & d & 0 & -c & 0 & b \\ 0 & -d & 0 & b & 0 & -e & 0 & -c \\ 0 & -c & 0 & e & 0 & b & 0 & d \\ 0 & -b & 0 & -c & 0 & -d & 0 & -e \end{bmatrix}$

and $P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

The computational complexity of P_0 is only 8 additions. To reduce the complexity of I_0, we use permutation techniques by performing the operations:

$$I_0 = \tilde{I} \cdot P_C \tag{3.9}$$

Here $P_C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

and $\tilde{I} = \begin{bmatrix} a & f & a & g & 0 & 0 & 0 & 0 \\ a & g & -a & -f & 0 & 0 & 0 & 0 \\ a & -g & -a & f & 0 & 0 & 0 & 0 \\ a & -f & a & -g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -e & d & -c & b \\ 0 & 0 & 0 & 0 & -d & b & -e & -c \\ 0 & 0 & 0 & 0 & -c & e & b & d \\ 0 & 0 & 0 & 0 & -b & -c & -d & -e \end{bmatrix}$

There is no computational cost for P_C as it only permutes the input data set (just needs rewiring). $\tilde{I}$ can be further decomposed by the "direct sum" operation. The direct sum is an operator (with a symbol of '$\oplus$') that can split a matrix into small sub matrices by retaining non-zero coefficients and eliminating zero coefficients. In the following, we show how $\tilde{I}$ is split into two 4×4 sub-matrices, $\tilde{I}_{00}$ and $\tilde{I}_{11}$ by the direct sum operation ('$\oplus$'):

$$\tilde{I} = \tilde{I}_{00} \oplus \tilde{I}_{11} \tag{3.10}$$

Where $\tilde{I}_{00} = \begin{bmatrix} a & f & a & g \\ a & g & -a & -f \\ a & -g & -a & f \\ a & -f & a & -g \end{bmatrix}$ and $\tilde{I}_{11} = \begin{bmatrix} -e & d & -c & b \\ -d & b & -e & -c \\ -c & e & b & d \\ -b & -c & -d & -e \end{bmatrix}$

It is important to note here that $\tilde{I}_{00}$ has the same structure as $I_{4\times4}$ (of Eqn. (3.1)). As a result, we will be able to reuse vast resources in the computation of $\tilde{I}_{00}$ from already developed hardware units of $I_{4\times4}$. Thus, Eqn. (3.8) can be re-expressed as:

$$I_{8\times8} = P_0 . (\tilde{I}_{00} \oplus \tilde{I}_{11}) . P_C \tag{3.11}$$

Eqn. (3.11) forms the general expression of Eqn. (3.7). We will use $\tilde{I}_{00}$ and $\tilde{I}_{11}$ as the basic building blocks to compute other 8x8 IDCTs. Since, the coefficients in $\tilde{I}_{00}$ and $\tilde{I}_{11}$ are fixed, they can be independently implemented, enabling fast computation. To achieve maximum sharing, we will implement P_0 and P_C only once and reuse them for all other 8x8 transforms.

In the following section, we show how Eqn. (3.11) can be applied to different Int-IDCT matrices. Another new feature of the proposed scheme is that we take the advantage of the similarity in matrix operation to further optimize the implementation. First of all, we apply Eqn. (3.11) to efficiently implement the transformation matrix of AVS-P2. Based on it and the generalized structure, we develop the matrix of VC-1 so that we can share as many units (from AVS-P2) as possible. Next, we develop the IDCT matrix of H.264 based on the same principle (decompose and share from AVS-P2 and VC-1). In this stage, we are able to achieve the maximum sharing as it will be shown later (in Section 3.3.4) that the implementation of H.264 does not cost any extra hardware. Finally, we develop the IDCT of HEVC by further decomposing and reusing the units already implemented (with a minimum addition of extra units).

3.3.2 Matrix decomposition for 8x8 AVS-P2

First we construct A_{8x8} (from Eqn. (3.7) and Table 3-2) and then apply Eqn. (3.11) to compute the 4×4 sub-matrices, $\tilde{A}_{00}$ and $\tilde{A}_{11}$. We then right shift $\tilde{A}_{00}$ by three bits and decompose it as follows:

$$\frac{\tilde{A}_{00}}{8}=\begin{bmatrix} 1 & \frac{5}{4} & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -\frac{5}{4} \\ 1 & -\frac{1}{2} & -1 & \frac{5}{4} \\ 1 & -\frac{5}{4} & 1 & -\frac{1}{2} \end{bmatrix}=A_1\cdot A_{2a\_8p} \tag{3.12}$$

Where $A_{2a\_8p} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & \frac{5}{4} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & -\frac{5}{4} \end{bmatrix}$

In Eqn. (3.12), A_1 is reused from Eqn. (3.2). To compute A_{2a_8p} from the shared the architecture of A_2 / A_{2h_4p}, we add an input pin to the multiplexer. The new factor $(5/4)\cdot x$ is implemented as $(1+1/4).x$. Thus in Eqn. (3.12), the total computational cost is 2 addition operations. In similar way, we can decompose $\tilde{A}_{11}$ as shown below:

$$\frac{\tilde{A}_{11}}{4} = \begin{bmatrix} -\frac{1}{2} & \frac{3}{2} & -\frac{9}{4} & \frac{5}{2} \\ -\frac{3}{2} & \frac{5}{2} & -\frac{1}{2} & -\frac{9}{4} \\ -\frac{9}{4} & \frac{1}{2} & \frac{5}{2} & \frac{3}{2} \\ -\frac{5}{2} & -\frac{9}{4} & -\frac{3}{2} & -\frac{1}{2} \end{bmatrix} = A_3 \cdot A_4 \qquad (3.13)$$

Where $A_3 = \begin{bmatrix} -1 & 0 & \frac{3}{2} & 1 \\ 0 & 1 & 1 & -\frac{3}{2} \\ -\frac{3}{2} & 1 & -1 & 0 \\ -1 & -\frac{3}{2} & 0 & -1 \end{bmatrix}$ and $A_4 = \begin{bmatrix} \frac{3}{2} & 0 & 0 & -1 \\ 0 & \frac{3}{2} & 1 & 0 \\ 0 & 1 & -\frac{3}{2} & 0 \\ 1 & 0 & 0 & \frac{3}{2} \end{bmatrix}$

For both A_3 and A_4, the coefficient (3/2) can be shared and the cost is: 12 additions and 4 shift operations for A_3; 8 additions and 4 shift operations for A_4. From Eqn. (3.12)-(3.13), we can summarize the final expression of the 8×8 IDCT for AVS-P7 as:

$$A_{8\times 8} = 4 \cdot P_0 \cdot \left[\left(A_1 \cdot 2A_{2a\_8p} \right) \oplus \left(A_3 \cdot A_4 \right) \right] \cdot P_C \tag{3.14}$$

Hence, the total computational cost to implement $A_{8\times 8}$ (including the cost of P_0 and P_C) is 30 additions and 12 shift operations.

In the next section, we will apply Eqn. (3.11) to VC-1 and subsequently decompose the matrix in a way so that we can reuse the units already developed for the AVS-P7 (from Eqn. (3.14)).

3.3.3 Matrix decomposition for 8x8 VC-1

We follow the same principles, as discussed in Eqn. (3.12)-(3.13), to decompose the 8x8 Int-IDCT for the VC-1.

$$V_{8\times 8} = P_0 \cdot \tilde{V} \cdot P_C \tag{3.15}$$

$$\text{Where, } \tilde{V} = \tilde{V}_{00} \oplus \tilde{V}_{11} \tag{3.16}$$

Now considering the symmetric property and the coefficient distribution patterns between $\tilde{A}_{00}/8$ (in Eqn. (3.12)) and $\tilde{V}_{00}/8$, we decompose $\tilde{V}_{00}/8$ as:

$$\frac{\tilde{V}_{00}}{8} = \begin{bmatrix} \frac{3}{2} & 2 & \frac{3}{2} & \frac{3}{4} \\ \frac{3}{2} & \frac{3}{4} & -\frac{3}{2} & -2 \\ \frac{3}{2} & -\frac{3}{4} & -\frac{3}{2} & 2 \\ \frac{3}{2} & -2 & \frac{3}{2} & -\frac{3}{4} \end{bmatrix} = A_1 \cdot V_{2\_8p} \tag{3.17}$$

$$\text{Where } V_{2\_4p} = \begin{bmatrix} \frac{3}{2} & 0 & \frac{3}{2} & 0 \\ \frac{3}{2} & 0 & -\frac{3}{2} & 0 \\ 0 & 2 & 0 & \frac{3}{4} \\ 0 & \frac{3}{4} & 0 & -2 \end{bmatrix} = 2A_2 - V_{3\_8p} \tag{3.18}$$

$$\text{and } V_{3\_8p} = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 & -\frac{1}{2} \end{bmatrix}$$

From Eqn. (3.18), we can re-express Eqn. (3.17) as:

$$\frac{\tilde{V}_{00}}{8} = A_1 \cdot \left(2A_2 - V_{3\_8p}\right) \tag{3.19}$$

In Eqn. (3.19), V_{3_8p} is the only new component. It has similar structure to V_3 of (2), except for the 2nd and 4th columns. Hence to calculate V_{3_8p}, we have added four multiplexers to the hardware unit of V_3. So in (18), the total computational cost is 4 shift operations only.

Now, based on our careful observation between the computational similarities between $\tilde{A}_{11}/4$ (in (12)) and $\tilde{V}_{11}/8$, we devise the decomposition scheme of $\tilde{V}_{11}/8$ as:

$$\frac{\tilde{V}_{11}}{8} = \begin{bmatrix} -\frac{1}{2} & \frac{9}{8} & -\frac{15}{8} & 2 \\ -\frac{9}{8} & 2 & -\frac{1}{2} & -\frac{15}{8} \\ -\frac{15}{8} & \frac{1}{2} & 2 & \frac{9}{8} \\ -2 & -\frac{15}{8} & -\frac{9}{8} & -\frac{1}{2} \end{bmatrix} = V_4 \cdot A_{4v} \tag{3.20}$$

Where, $V_4 = A_3 + A_{3V}$ (3.21)

$$A_{4v} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & -1 \\ 0 & \frac{1}{4} & 1 & 0 \\ 0 & 1 & -\frac{1}{4} & 0 \\ 1 & 0 & 0 & \frac{1}{4} \end{bmatrix} \text{ and } A_{3v} = \begin{bmatrix} -1 & -\frac{3}{2} & 0 & -1 \\ \frac{3}{2} & -1 & 1 & 0 \\ 0 & 1 & 1 & -\frac{3}{2} \\ 1 & 0 & -\frac{3}{2} & -1 \end{bmatrix}$$

By substituting Eqn. (3.21) in Eqn. (3.20), $\tilde{V}_{11}/8$ can be expressed as:

$$\frac{\tilde{V}_{11}}{8} = (A_3 + A_{3V}) \cdot A_{4v} \tag{3.22}$$

We note that, A_{4v} in Eqn. (3.20) is structurally similar to A_4 in Eqn. (3.13), except for the change in the diagonal coefficients. So we only need to implement it; the rest is shared from the architecture of A_4. We do so by adding four multiplexers for the four left diagonal elements of A_4 matrix. Then according to Eqn. (3.21), we reuse A_3 to compute V_4. As the new matrix A_{3v} can be derived from A_3 by rearranging the rows and changing the polarity of some input bits, we share it from the design of A_3 by adding 4 multiplexers only. Finally the expression of $\tilde{V}_{00}$ and $\tilde{V}_{11}$ from Eqn. (3.19) and Eqn. (3.22) are substituted in Eqn. (3.15) to get the final expression of the IDCT for VC-1:

$$V_{8\times 8} = 8 \cdot P_0 \cdot \left\{ \left[A_1 \cdot \left(2A_2 - V_{3\_8p} \right) \right] \oplus \left[(A_3 + A_{3V}) \cdot A_{4v} \right] \right\} \cdot P_C \qquad (3.23)$$

It is seen from Eqn. (3.23) that to implement $V_{8\times 8}$, the only required new unit is V_3; the rests are shared from the implementation of AVS-P2 (from Eqn. (3.14)). So, the total computational cost for VC-1 (excluding the cost of P_0 and P_C) is 4 addition and 8 shift operations.

3.3.4 Matrix decomposition for 8x8 H.264/AVC

Following similar procedure illustrated in the two previous sections, we can simplify the 8x8 transformation matrix for H.264/AVC as shown below:

$$H_{8\times 8} = P_0 \cdot \tilde{H} \cdot P_C \qquad (3.24)$$

$$\text{Where, } \tilde{H} = \tilde{H}_{00} \oplus \tilde{H}_{11} \qquad (3.25)$$

In order to ensure the maximum unit sharing, we decompose $\tilde{H}_{00} / 8$ as below:

$$\frac{\tilde{H}_{00}}{8} = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} = A_1 \cdot A_{2h\_8p} \qquad (3.26)$$

Where $A_{2h\_8p} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & -1 \end{bmatrix}$

In Eqn. (3.26), A_1 is directly reused from Eqn. (3.14). To compute A_{2h_8p} from the shared architecture of $A_2 / A_{2h\_4p} / A_{2a\_8p}$, we simply add another input to the multiplexers. So there is no additional cost in terms of adders and shifters to compute $\tilde{H}_{00} / 8$. Similarly, we can decompose $\tilde{H}_{11} / 8$ as:

$$\frac{\tilde{H}_{11}}{8} = \begin{bmatrix} -\frac{3}{8} & \frac{3}{4} & -\frac{5}{4} & \frac{3}{2} \\ -\frac{3}{4} & \frac{3}{2} & -\frac{3}{8} & -\frac{5}{4} \\ -\frac{5}{4} & \frac{3}{8} & \frac{3}{2} & \frac{3}{4} \\ -\frac{3}{2} & -\frac{5}{4} & -\frac{3}{4} & -\frac{3}{8} \end{bmatrix} = A_{3h} \cdot A_{4v} \tag{3.27}$$

Where $A_{3h} = \begin{bmatrix} -\frac{3}{2} & -1 & 1 & 0 \\ 1 & 0 & \frac{3}{2} & -1 \\ -1 & \frac{3}{2} & 0 & -1 \\ 0 & -1 & -1 & -\frac{3}{2} \end{bmatrix}$

Here A_{4v} is directly reused from Eqn. (3.23) and we share A_{3h} from the architecture of A_3. For this share we do not even need to use any multiplexer, because we have already done so while sharing A_{3v} from A_3 in Section 3.2.1. The final expression of the 8x8 IDCT for H.264 (with all shared units) can be summarized as follows:

$$H_{8\times 8} = 8 \cdot P_0 \cdot \left\{ \left[A_1 \cdot A_{2h} \right] \oplus \left[A_{3h} \cdot A_{4v} \right] \right\} \cdot P_C \tag{3.28}$$

It is interesting to note that all terms in Eqn. (3.28) are implemented from the terms of Eqn. (3.14) and (3.23); so, in the proposed scheme, there is no additional cost to implement the IDCT for H.264 which results in significant hardware savings.

3.3.5 Matrix decomposition for 8x8 HEVC

In this section, we develop the transformation matrix for the HEVC based on the principles described before. The 8×8 matrix can be decomposed as:

$$HV_{8\times 8} = P_0 \cdot \widetilde{HV} \cdot P_C \tag{3.29}$$

$$\text{Where, } \widetilde{HV} = \widetilde{HV_{00}} \oplus \widetilde{HV_{11}} \tag{3.30}$$

$$\text{and } \widetilde{HV_{00}} = \begin{bmatrix} 64 & 83 & 64 & 36 \\ 64 & 36 & -64 & -83 \\ 64 & -36 & -64 & 83 \\ 64 & -83 & 64 & -36 \end{bmatrix} = HV_{4\times 4} \tag{3.31}$$

From Table 3-1 and Table 3-2 we can see that the a, f, g coefficients of 4x4 and 8x8 HEVC transform matrices are same. Therefore, $\widetilde{HV_{00}}$ and $HV_{4\times 4}$ (in Eqn. (3.6)) are same. So there is no additional cost to compute $\widetilde{HV_{00}}$ in Eqn. (3.31). We further decompose $\widetilde{HV_{11}} / 4$ as:

$$\frac{\widetilde{HV_{11}}}{4} = \begin{bmatrix} -\frac{9}{2} & \frac{25}{2} & -\frac{75}{4} & \frac{89}{4} \\ -\frac{25}{2} & \frac{89}{4} & -\frac{9}{2} & -\frac{75}{4} \\ -\frac{75}{4} & \frac{9}{2} & \frac{89}{4} & \frac{25}{2} \\ -\frac{89}{4} & -\frac{75}{4} & -\frac{25}{2} & -\frac{9}{2} \end{bmatrix} = (8A_3 + HV_2) \cdot A_{4HV} \qquad (3.32)$$

Where $HV_2 = \begin{bmatrix} \frac{7}{4} & \frac{5}{4} & -2 & 0 \\ -\frac{5}{4} & 0 & -\frac{7}{4} & 2 \\ 2 & -\frac{7}{4} & 0 & \frac{5}{4} \\ 0 & 2 & \frac{5}{4} & \frac{7}{4} \end{bmatrix}$ and $A_{4HV} = \begin{bmatrix} 2 & 0 & 0 & -1 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & -2 & 0 \\ 1 & 0 & 0 & 2 \end{bmatrix}$

Combining Eqn. (3.29)-(3.32), we compute the proposed 8×8 Int-IDCT for HEVC as given below:

$$HV_{8\times8} = 4 \cdot P_0 \cdot \left\{ \left[A_1 . (16A_2 + HV_1) \right] \oplus \left[(8A_3 + HV_2) \cdot A_{4HV} \right] \right\} \cdot P_C \qquad (3.33)$$

In Eqn. (3.32), the only new matrix HV_2 will be implemented and A_{4HV} will be shared from the common resource of A_4 / A_{4V} by adding another extra input to all multiplexer units. The rest will be shared from Eqn. (3.6) and (3.14). So the total computational cost to implement $HV_{8\times8}$ in the proposed design is 16 addition and 12 shift operations (excluding the cost of P_0 and P_C).

In summary, the proposed 8x8 multi IDCT design costs 50 additions and 32 shift operations. And the overall cost of our design is 76 addition and 64 shift operations to perform both 4x4 and 8x8 inverse transformations of four video standards.

3.4 Extensibility to larger transform units of HEVC

The proposed decompose and share algorithm, as described previous sections, can be extended for larger transform units of HEVC (such as, 16x16 and 32x32 as defined in [52]). In that case, Eqns. (3.8)-(3.11) need to be modified accordingly to accommodate larger matrices. To implement the 16x16 using the proposed shared scheme, we will require 38 more adders, while for 32x32 implementation, 76 additional adders will be required. Since, other existing codecs (H.264, VC-1, and AVS) do not support larger transforms, we do not include the detailed implementation of larger transform units (of HEVC) in this thesis.

CHAPTER 4

HARDWARE IMPLEMENTATION

4.1 Introduction

The proposed hardware can perform the Int-IDCT operation for all four video standards. The overall block diagram of our proposed scheme is shown in Figure 4.1. A user (or any external source) can choose the type of video codec and size of transform matrix by a selection pin (*sel*). Only In-IDCT operation of one video codec and its associated computational units are activated at a time by the control unit; the rest are disabled. To implement this multi-codec design we have shared the entire hardware unit of the 4×4 matrices, instead of sharing individual adders, shifters, or other factors (as done in [14]). It ensures maximum reduction of hardware cost in our design. In the proposed architecture, we need to store data for short duration of time inside the functional units; as a result, we have used flip-flops instead of an array of memory blocks to store intermittent data.

The block diagram of our proposed architecture is shown in Figure 4.1. According to this diagram, we can split the entire architecture into the following major blocks:

I. Serial to Parallel Converter (S2PC)

II. P_0 and A_1 blocks

III. Block – b1

IV. Block – b2

V. Block – b3

VI. P_C block

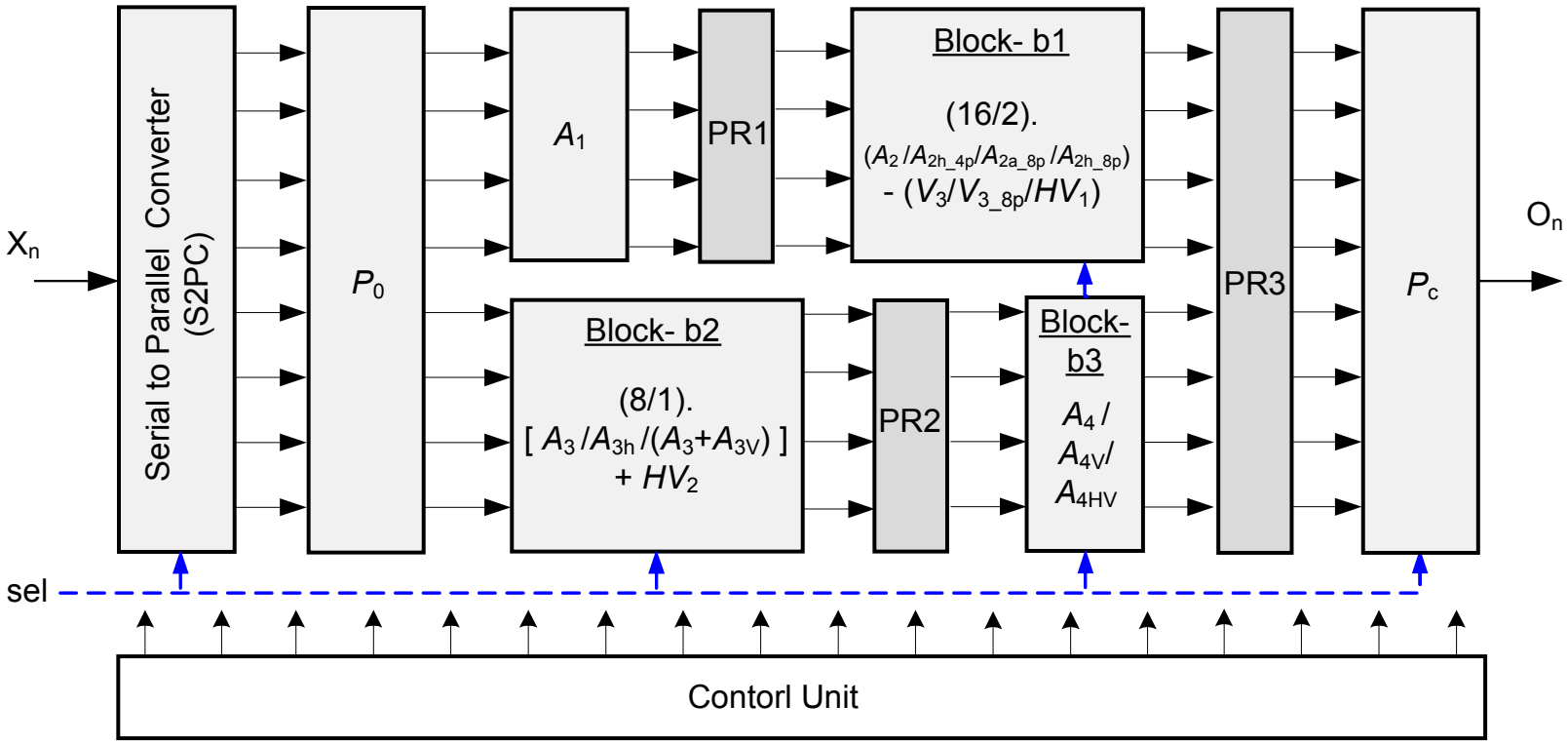

Figure 4.1. Block diagram of the proposed architecture

All these blocks execute operations according to the algorithm. The design employs a 3-stage pipeline to improve operational frequency and maintain data synchronism; the pipeline registers (PR1, PR2, and PR3) are also shown in Figure 4.1.

In the following sections, we have briefly described the hardware architecture of all the major blocks.

4.2 Architecture of Serial to Parallel Converter (S2PC)

The proposed multi-codec transform unit receives input one by one from the previous process at every clock cycle. As the transform operation cannot be initiated unless a complete input row available, hence during the 8x8 transform operation the S2PC block stores the input one by one into eight registers (X_0, X_1,.., X_7) in 8 clock cycles, and at the 9^{th} cycle all stored input samples are sent to next block P_0. Similarly, during the 4x4 transform operation the four inputs are stored at the top four registers and every 5^{th} cycle they will be sent to P_0 block. The last four registers always hold '0' during the 4x4 transform. Figure 4.2 shows the hardware architecture of the S2PC block.

Here the S2PC block apparently functions like a memory unit as it stores the rows of the input matrix inside the 8 registers as shown in Figure 4.2. That is the reason the proposed design does not require additional memory architecture.

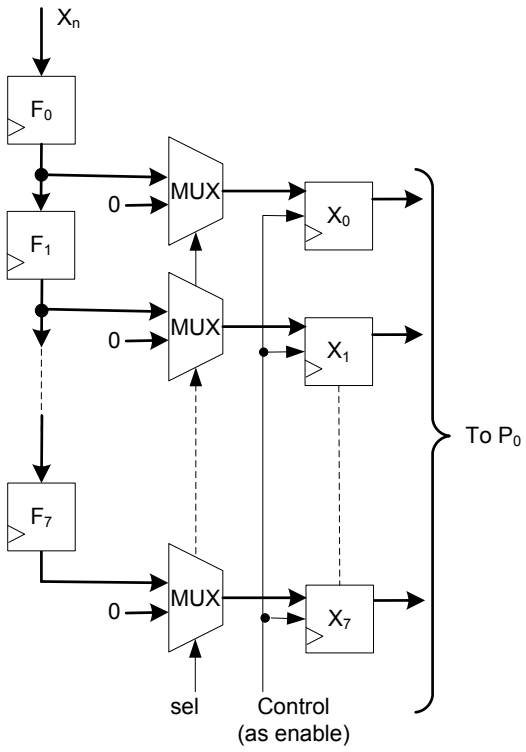

Figure 4.2. Hardware architecture of S2PC block

4.3 Architecture of P_0 and A_1 blocks

The hardware blocks 'P_0' and 'A_1' perform the same operations as the matrix P_0 and A_1 function at the previous chapter. We can see from Figure 4.1 that the P_0 block splits the 8x8 decomposition process into two independent 4x4 processes. Since these 4x4 processes work concurrently, the design throughput is increased. The detail hardware architecture of P_0 and A_1 blocks are illustrated in Figure 4.3.

From the figures it is seen that a series of addition and subtraction operations is performed inside these two blocks according to the matrices structure of P_0 and A_1 of Chapter-3. During the 4x4 transform operation, the select pin will bypass the input data set from the P_0 block as described in the previous chapter.

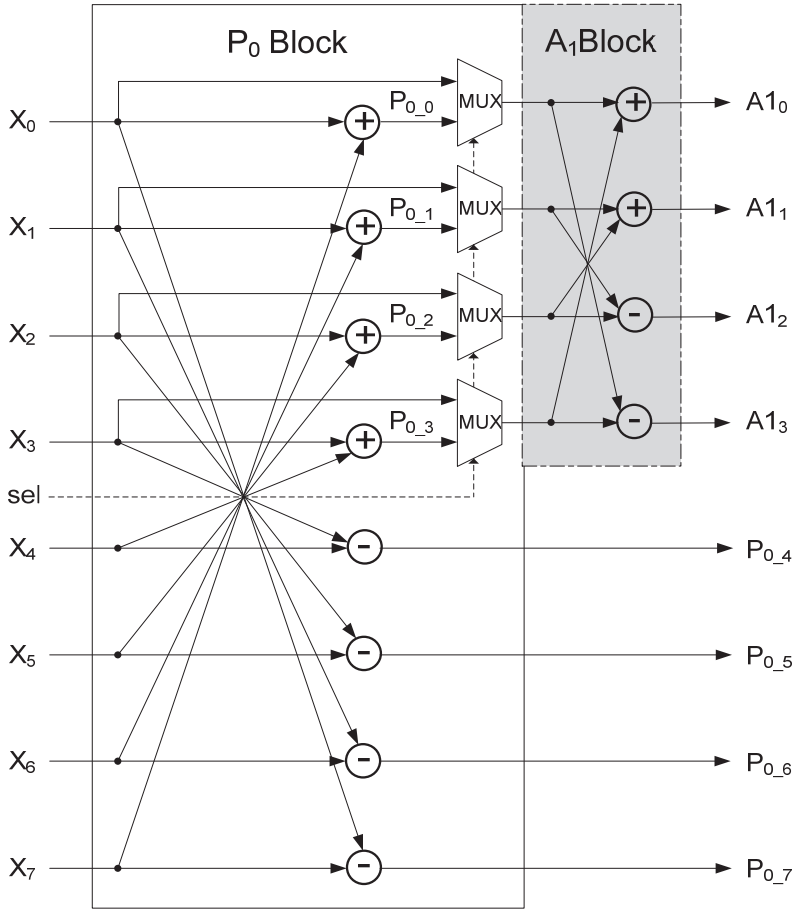

Figure 4.3. Hardware architecture of P_0 and A_1 blocks.

4.4 Architecture of Block-b1

The hardware architecture of 'Block-b1' is illustrated in Figure 4.4. The mathematical operation performed inside this block is (16/2). [(A_2 /A_{2h_4p}/A_{2a_8p} /A_{2h_8p}) - (V_3/V_{3_8p}/HV_1)] . Here the term 'A_2 /A_{2h_4p}/A_{2a_8p} /A_{2h_8p}' means the shared architecture for the matrices: A_2 , A_{2h_4p}, A_{2a_8p} and A_{2h_8p} of Chapter 3. According to our algorithm we utilize the structural similarities among these matrices to design this shared hardware. Depending on the select pin and control signal, at a certain times this shared design operates like a 4x4 matrix of A_2 or A_{2h_4p} or A_{2a_8p} or A_{2h_8p} of Chapter 3. The internal hardware share strategies of 'A_2 /A_{2h_4p}/A_{2a_8p} /A_{2h_8p}' architecture are shown in Figure 4.5.

Similarly, the detail of shared hardware '$V_3/V_{3\_8p}/HV_1$' is illustrated in Figure 4.6. In these figures, '<<n' and '>>n' symbols mean n-bit left shift and n-bit right shift respectively of the corresponding input data.

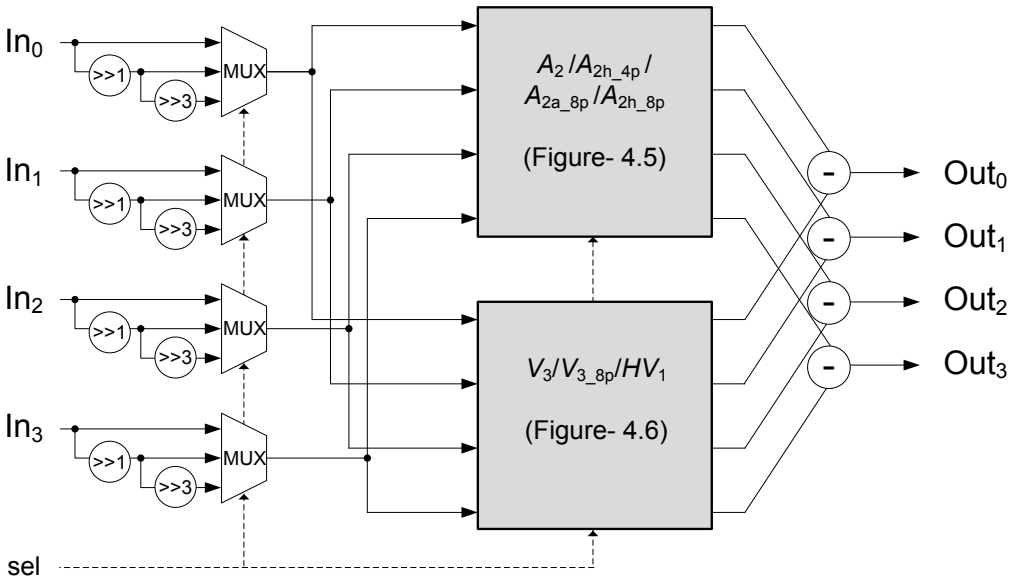

Figure 4.4. Hardware architecture of Block-b1

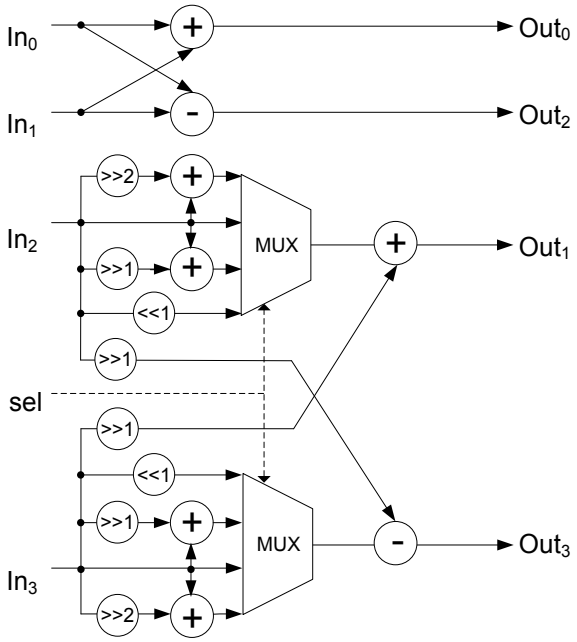

Figure 4.5. Shared hardware for $A_2/A_{2h\_4p}/A_{2a\_8p}/A_{2h\_8p}$

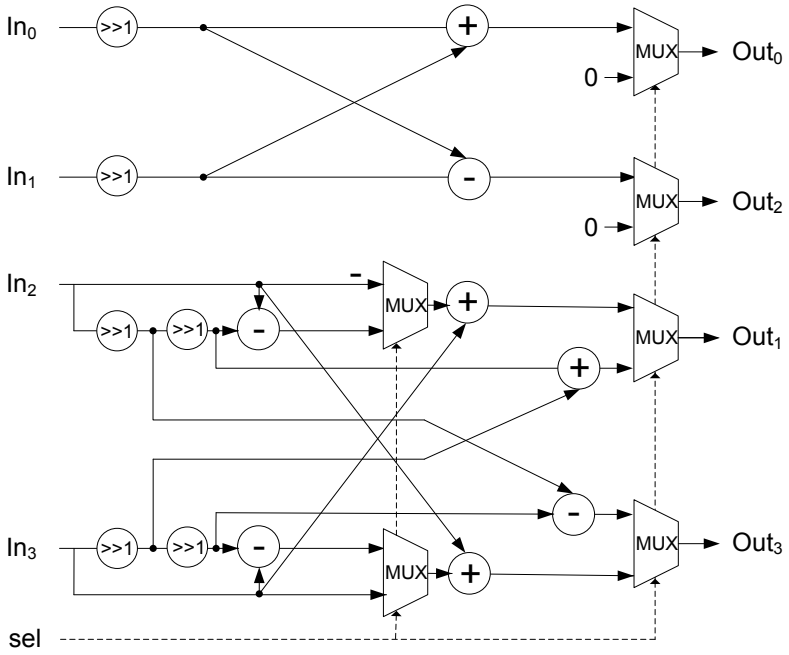

Figure 4.6. Shared hardware for $V_3/V_{3\_8p}/HV_1$

At different stages in these designs, several multiplexers are used to ensure proper computation of the IDCT in operation. These multiplexers are controlled by the control unit.

4.5 Architecture of Block-b2

The mathematical operation performed by Block-b2 is (8/1). [$A_3 / A_{3h} / (A_3 + A_{3V})$] + HV_2. All the matrices of this expression are defined at the previous chapter. Here the shared hardware '$A_3 / A_{3h} / (A_3 + A_{3V})$' also function like those of Block-b1. The detail architecture of Block-b2 and its shared unit are shown in Figure 4.7 and Figure 4.8 respectively.

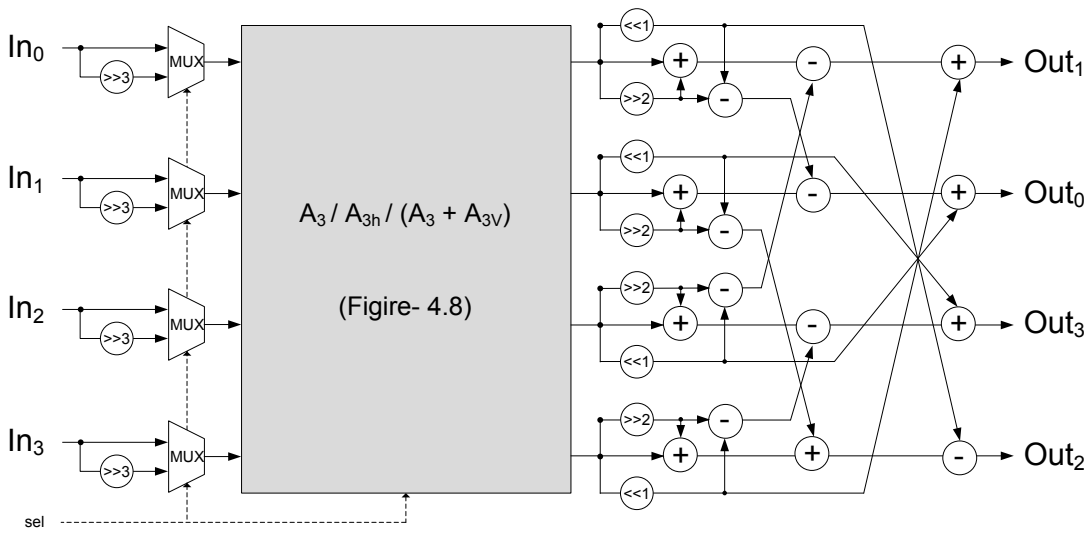

Figure 4.7. Hardware architecture of Block-b2

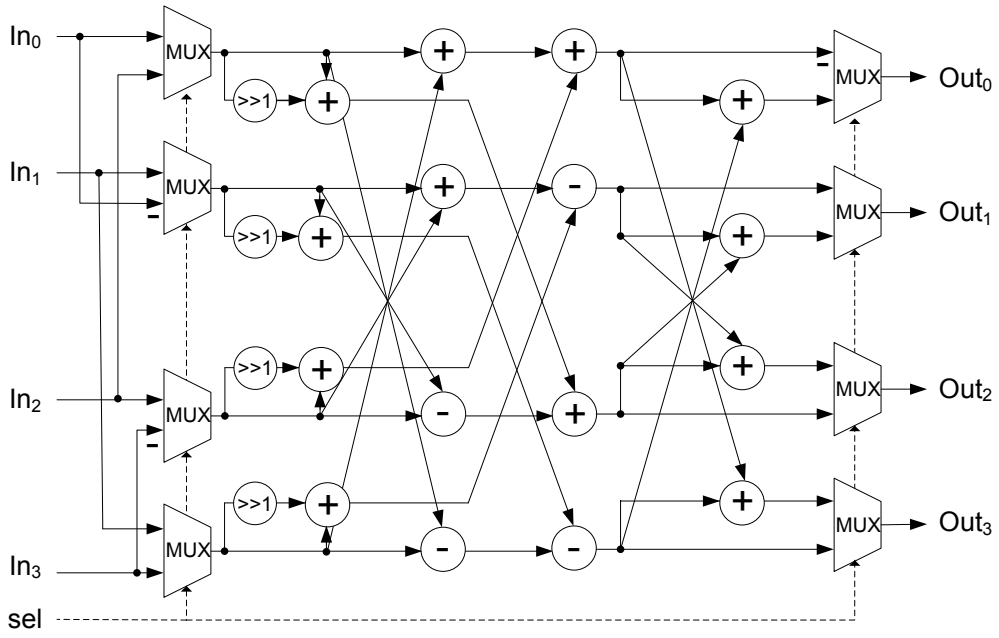

Figure 4.8. Shared hardware for A_3 /A_{3h} /(A_3+A_{3V})

4.6 Architecture of Block-b3

The function performed by Block-b3 is 'A_4 /A_{4V}/A_{4HV}'. That means Block-b3 is a shared architecture that can function like matrix A_4 or A_{4V} or A_{4HV} depending on the select pin and control signal. The hardware detail of this block is shown in Figure 4.9.

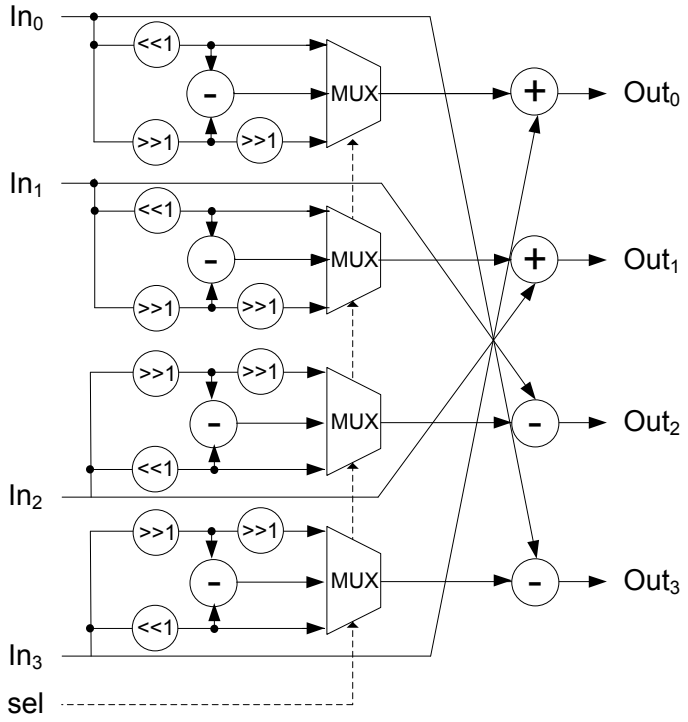

Figure 4.9. Hardware architecture of A_4 /A_{4V}/A_{4HV} (Block-b3)

4.7 Architecture of P_c block

It is seen from Figure 4.10 that the P_C block combines two different sets of data from Block-b2 and Block-b3 and generates one output. The function of this block is the same as multiply the input data set with P_C matrix of chapter-3. As the P_C matrix has no computational cost, we do not need to perform any addition or subtraction operation to implement the P_C block.

Inside this block, the inputs of Block-b2 are stored in the first four registers (R_{00}, R_{01}, R_{02}, R_{03}) and the inputs of Block-b2 are stored in the last four registers (R_{10}, R_{11}, R_{12}, R_{13}). Then according to the control signal, in every clock cycle, one output is generated through O_n.

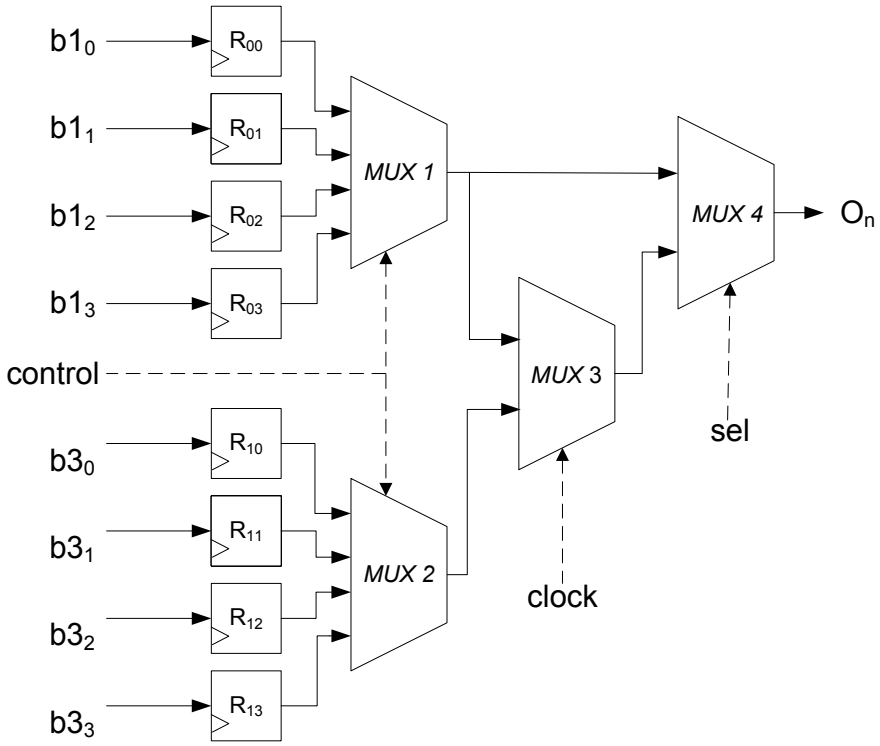

Figure 4.10. Hardware architecture of P_c block.

Chapter 5

PERFORMANCE ASSESSMENT

5.1 Introduction

The proposed architecture is synthesized on both Xilinx Virtex4 LX60 FPGA and 0.18μm CMOS technology. In this chapter, we analyze the synthesis results in term of hardware resources. Next, the latency and hardware sharing efficiency of the proposed design are measured. Finally, the hardware cost, transform features and video frame decoding capability of our multi-codec architecture are compared with other existing design.

5.2 Analysis of synthesis result

First the proposed design is modeled in Verilog HDL and then the operation is verified using a Xilinx Virtex4 LX60 FPGA. It costs 2,252 Lookup Tables (LUT) and 950 bit registers. In Table 5-1, we present the breakdown of the cost of all major components of the design (as illustrated in Figure.4.1). It can be observed from this table that Block-b2 requires significantly higher LUTs compare to others, which is reasonable considering the fact that it performs good amount of computations of the full design.

The design is later synthesized using 0.18μm CMOS technology. The architecture costs 39.4k gates and 12.27k standard cells with a maximum operating frequency of 200.8MHz. The synthesis result of the entire architecture using Xilinx Virtex4 LX60 FPGA and 0.18μm CMOS technology are summerized in Table 5-2.

Table 5-1: Breakdown of hardware cost

Block name	Adder	LUT	Area % of LUT	Register
S2PC	0	24	1.06%	72
P_0	8	103	4.58%	96
A_1	4	57	2.53%	52
Block- b1	24	501	22.24%	86
Block- b2	32	1166	51.77%	345
Block- b3	8	359	15.95%	94
*P*c	0	20	0.89%	200
Control unit	0	22	0.98%	5
Total	**76**	**2252**	**100%**	**950**

Table 5-2: Summary of synthesis report

0.18 um CMOS technology	**Power**	29.9mW
	Total Std. Cell	12272
	Area	0.3544 Sq.mm
	Total gate count	39.4K
	Frequency	200.8MHz
Xilinx Virtex4 LX60	**Lookup table**	2243
	Max. Frequency	208.4MHz

From Table 5-2, we can see that the 0.18μm CMOS standard cell technology achieved slightly less frequency than the Virtex4 FPGA. The reason is that, this particular FPGA runs by an optimized 0.09μm CMOS processor which has smaller area and higher frequency than its predecessor (0.18μm technology) [63][64].

5.3 Measurement of latency

Our design receives 1 pixel/cycle as input. So the S2PC block of Figure 4.1 requires 8 (or 4 cycles) to fully receive one data vector of an 8x8 (or 4x4) input matrix. It requires 3 more cycles for three pipeline stages. As a result, the decoding of one row takes 11 (3+8) cycles for 8x8 IDCT and 7 (3+4) cycles for 4x4 IDCT. Our proposed design can be easily extended for two dimensional IDCT by adding a transpose unit. For two dimensional IDCT, it requires additional 64 (for 8x8) or 16 cycles (for 4x4) to store data in the transpose memory. Hence the latency is 86 (11+64+11) cycles for a full 2D 8x8 operation and is 30 (7+16+7) cycles for 2D 4x4 operation.

5.4 Measurement of hardware sharing efficiency

In VLSI design, the cost of hardware linearly depends on the number of arithmetic units used, such as adders and shifters. The more these computational units are used inside a design, the more hardware is required for implementation; which ultimately increase the chip area. Hence, in many System-On-a-Chip (SOC) designs, the number of arithmetic units is used as index to compare hardware resource.

From Chapter 3, we know that the proposed sharing algorithm ensures significant resource reduction. In order to demonstrate the sharing efficiency at the hardware level, we have compared the adder count of our design with the 4x4 and 8x8 standalone Int-IDCT matrices of three standards: AVS, VC-1 and H.264/AVC (as presented in [16]). The results are shown in Figure 5.1. As till now, there is no implementation of the 4x4 and 8x8 Int-IDCT of HEVC; so, we have implemented it separately for the sake of better comparison. We can see from Figure 5.1 that total 145 adders are required to implement these four transforms without sharing. The proposed shared design can compute all of

them with 47.6% less adders. Moreover, the savings achieved in individual standards due to the sharing are marked by the dotted line in Figure 5.1.

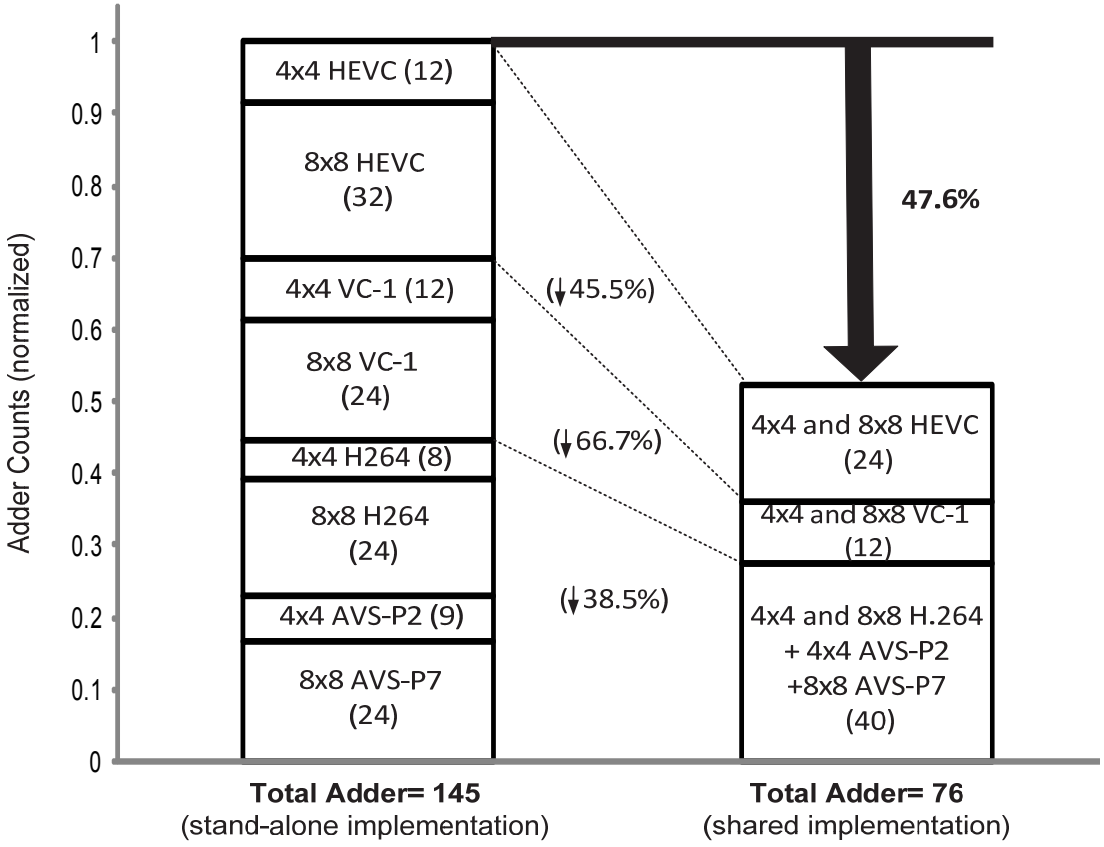

Figure 5.1. Cost of the proposed scheme: Standalone vs. Cost-shared

It is important to note that, 4x4 AVS-P7, 8x8 AVS-P2, 4x4 H.264 and 8x8 H.264 combined together cost only 40 adders (compared to 65 for standalone implementations). The cost of implementing shift operation is considered insignificant in the calculation. As mentioned earlier, we have presented in this work an efficient implementation of the inverse DCT; however, the cost to implement the forward DCT remains the same as it uses the same basis coefficients inside the transpose matrices.

5.5 Comparison of hardware cost with existing designs

In Table 5-3, we compare the cost of the proposed scheme (in terms of adder and shifter) with available existing designs in the literature. None of the designs in this table supports HEVC (which is computationally expensive due to large matrix parameters as shown in Table 3-1 and 3-2). Although, the design in [16] costs fewer adders, it is shown later that the proposed scheme outperforms it in decoding capacity. Besides, it does not support 4x4 transforms. Considering the fact that, the proposed architecture can decode both 4x4 and 8x8 Int-IDCT for four video codecs, it consumes the least number of adders compared to all other designs.

Table 5-3: Comparison of the cost of adders and shifters

Supported Codecs	**# of adders**	**# of shifters**
8x8 JPEG + 8x8 MPEG-2/4 + 4x4 H.264 + 8x8 VC-1 in [12]	112	--
8x8 JPEG + 8x8 MPEG-2/4 + 4x4 H.264 +4x4 & 8x8 VC-1 in [14]	70	--
4x4 & 8x8 H.264 + 8x8 VC-1 in [18]	76	28
8x8 MPEG-2/4 + 8x8 H.264 + 8x8 VC-1+ 8x8 AVS in [17]	76	--
8x8 JPEG+ 8x8 MPEG-2/4 + 8x8 H.264 + 8x8 VC-1+ 8x8 AVS in [16]	58	31
8x8 MPEG-2/4 + 4x4 & 8x8 H.264 + 4x4 & 8x8 VC-1 + 8x8 AVS [26]	76	--
Proposed – 4x4 & 8x8 H.264 + 4x4 & 8x8 VC-1 + 4x4 AVS-P2+ 8x8 AVS-P7 + 4x4 & 8x8 HEVC	76	64

'--' – No information

5.6 Comparison of transform features with existing designs

In Table 5-4, we have summarized the performance in terms of gate count, maximum working frequency, and supporting standards with other designs. As noted earlier, due to large magnitude of matrix coefficients of HEVC (compared with H.264, VC-1 and AVS as shown in Table 3-1 and 3-2), the proposed design costs more gates. However, from Table 5-4 we can see that it supports more modern video codecs (i.e., eight) and still has the highest operational frequency (200.8MHz). Only the design in [48] has a frequency closer to us, but it supports only H.264. Similarly, designs in [15] and [18] support only two codecs and accordingly cost less hardware than ours. Among other designs, [12], [13], [14], [16] and [17] are comparable to our design as they support as many as three codecs. While working at maximum capacity, the proposed design can process 200.8 million pixels/sec. As a result, the decoding capacity (with 4:4:4 luma-chroma sampling) of a full 1080p HD video frame (1920x1080) is 96.84Hz.

In Table 5-5, the decoding capability of the proposed approach is compared with the multi-transform designs which support at least three video codecs. In this comparison, we have used 4:2:0 luma-chroma sampling and recalculated the decoding capability. For example, in our work, the maximum achieved frame rate of 1080p video is = 200.8×10^6 / (1920 x 1080 + 2 x 960 x 540) = 64.56 ≈ 64 fps, which is the highest compared to all other designs listed in the table. Considering the current trends to use super resolution monitors, in Table 5-5 we have also compared the decoding capabilities for the Wide Quad eXtended Graphics Array (WQXGA, with resolution of 2560 ×1600 pixels).

Table 5-4: Comparison of multi-codec Int-IDCT architecture

Scheme	Technology	Gate count	Frequency (MHz)	Supported resolution		8x8 supported standards				4x4 supported standards			
				Full HD	WQXGA	H.264	VC-1	AVS-P7	HEVC	H.264	VC-1	AVS-P2	HEVC
Lee's [12]	0.13 µm	19.1k	136	Y	o	o	Y	o	o	Y	o	o	o
Kim's [13]	0.13 µm	30.9k	151	o	o	Y	Y	o	o	Y	Y	o	o
Qi's [14]	0.13 µm	18k	100	Y	o	o	Y	o	o	Y	Y	o	o
Lee's [15]	0.13 µm	10.5k	123	o	o	Y	Y	o	o	o	o	o	o
Wahid's [16]	0.18 µm	19.8k	194.7	Y	o	Y	Y	Y	o	o	o	o	o
Liu's [17]	0.13 µm	16.5k	110.8	--	--	Y	Y	Y	o	o	o	o	o
Fan's [18]	0.18 µm	7.14k	100	--	--	Y	Y	o	o	Y	o	o	o
Li's [48]	0.18 µm	13.7k	200	Y	o	Y	o	o	o	Y	o	o	o
Fan's [19]	0.18 µm	15.03k	100	--	--	o	Y	o	o	o	Y	o	o
Su's [55]	0.18 µm	8.12k	100	--	--	Y	o	o	o	Y	o	o	o
Wang's [26]	0.13 µm	23.06k	100	Y	o	Y	Y	Y	o	Y	Y	o	o
Proposed	0.18 µm	39.40k	200.8	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

'Y' – yes; 'o' – No; '--' – No information

Table 5-5: Comparison of decoding capability

Scheme	HD Resolution				WQXGA Resolution	
	1920 x 1080		1280 x 720		2560 x 1600	
	Time to transmit 1 frame (msec)	Frame per second (fps)	Time to transmit 1 frame (msec)	Frame per second (fps)	Time to transmit 1 frame (msec)	Frame per second (fps)
Lee's [12]	22.8	44	10.2	98	x	x
Kim's [13]	x	x	10.2	98	x	x
Qi's [14]	31.1	32	13.8	72	x	x
Wahid's [16]	16.7	60	7.1	140	x	x
Proposed	15.5	64	6.9	145	30.6	32

'x' – Not supported by the hardware

Thus, it can be seen that, the proposed design can not only decode AVS, H.264/AVC, VC-1 and HEVC videos, but also can maintain a higher operational frequency to meet the requirements of real time transmission. The target frame rate to real time transmit HD, full HD and QWXGA videos are set to 120, 60 and 30 fps respectively. From this performance analysis, the proposed scheme is found to be competitive as it can transmit the highest number of frames per second, and hence takes the least time to transmit one frame at a given resolution.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Summary of accomplishments

In this thesis, we have presented a generalized algorithm and a shared hardware architecture by using the symmetric property of the integer matrices and the matrix decomposition to compute 4x4 and 8x8 IDCT for four modern video codecs: H.264/AVC, VC-1, AVS and HEVC (in draft stage). The architecture is designed in such a way that it can accommodate changes (if any) in the final release of the HEVC. We first apply the generalized scheme to an AVS-based transform unit, and then gradually build the rest of the transform units on top of another to maximize the sharing.

It enables parallel operation and yields high throughput, which eventually helps meet the coding requirement of the high resolution video. The maximum achieved decoding capability for different resolution is: HD video (1280x720) @ 145 fps, full HD video (1920x1080) @ 64fps, and QWXGA video (2560x1600) @ 32fps.

The proposed algorithm also ensures maximum hardware sharing among the transform unit of four video codecs, which at the end reduce hardware resource of the full architecture. The performance analysis shows that this design requires 47.3% less hardware compared to an unshared design with similar transform capabilities. Overall, the architecture is suitable for low-cost implementation in modern multi-codec systems.

6.2 Recommendations for future work

In this section, some recommendations for future work are presented. These recommendations are beyond the scope of this thesis work, and left for the researchers who wish to continue the exploration. The recommendations are as follows:

I. In this work, matrix decomposition is used as sharing technique among the transform unit of different video codecs. Lee et al. proposed another efficient hardware sharing algorithm based on the novel concept of the delta coefficient matrix [12]. It also shares resources, such as adders and shifters, to optimize hardware cost [12]. This novel scheme can be further developed to implement a multi-codec reconfigurable transform architecture like the presented work with less hardware resource.

II. The implemented multi-codec unit only supports 4x4 and 8x8 block size transforms. However, the next generation video codec HEVC supports 16x16 and 32x32 block size [52], since larger transforms can bring higher performance in terms of energy compaction for large homogeneous areas. Thus as a continuation of this work, the 16x16 and 32x32 HEVC transform unit can be merged with our proposed design.

III. In this thesis, the DCT is used as the transform technique. It is also known as one dimensional DCT (1D-DCT). Recently, the 2D-DCT has also become very popular in multimedia industries. The computational steps of the 2D-DCT are exactly same as the 1D-DCT, except for an additional transpose operation [34]. Hence as a future challenge, the proposed design can be extended for the 2D-DCT by adding a transpose memory unit.

IV. In this work, we have developed and implemented a sharing algorithm for the transform unit of multiple video codecs. Similar approach can be taken to implement other units, such as quantization, motion compensation, deblocking filter, of different modern video codecs to save area and enhance performance.

REFERENCES

[1] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 3rd Ed. New Jersey: Prentice Hall, 2008, ch. 8. , pp. 525–538.

[2] *Course content of CME426: Multimedia* [Online]. Available: http://engrwww. usask.ca /classes/CME/462. Accessed on: April 21, 2012.

[3] S. Shrestha, "Hybrid DWT-DCT Algorithm for Image and Video Compression Applications," M.S. thesis, Dept. of ECE, Univ. of Sask., Saskatoon, SK, 2010.

[4] A. Rosenfeld and A. C. Kak, Digital picture processing. Academic Press Inc., 1982.

[5] N. C. Giri, Multivariate statistical inferences. Academic Press Inc., 1977.

[6] N. Ahmed, T. Natarajan, and K. Rao, "Discrete Cosine Transform," Computers, IEEE Transactions on, vol. C-23, no. 1, pp. 90–93, Jan 1974.

[7] K. R. Rao and P. Yip, Discrete Cosine Transform: Algorithms, Advantages, Applications. Boston: Academic Press, 1990.

[8] V. Bhaskaran and K. Konstantinides, Image and Video Compression Standards: Algorithm and Architectures, 2nd ed. Kulewer Academic Publisher, 1997.

[9] C. Wei, P. Hao, and Q. Shi, "Integer DCT-based Image Coding," Changjiang National Lab on Machine Perception, Peking University, Beijing, CHINA

[10] Y-J. Chen, S. Oraintara, and T. Nguyen, "Video compression using integer DCT," *in Proc. Int. Conf. Image Processing*, 2000.

[11] Joint Collaborative Team - Video Coding, "CE10: Core transform design for HEVC," JCTVC-G495, Geneva, Switzerland, 21-30 November, 2011.

[12] S. Lee and K. Cho, "Architecture of transform circuit for video decoder supporting multiple standards," *Electron. Lett.*, vol. 44, no. 4, pp. 274–275, Feb. 2008.

[13] S. Kim, H. Chang, S. Lee and K. Cho, "VLSI Design to Unify IDCT and IQ Circuit for Multistandard Video Decoder," in *Proc. ISIC Int. Symp. Prototyping*, pp. 328–331, 2008.

[14] H. Qi, Q. Huang and W. Gao, "A Low-Cost Very Large Scale Integration Architecture for Multistandard Inverse Transform," *IEEE Trans. Circuits and Systs. II,* vol. 57, no. 7, pp. 551–555, July 2010.

[15] S. Lee and K. Cho, "Circuit implementation for transform and quantization operations of H.264/MPEG-4/VC-1 video decoder," in *Proc. Int. Conf. Design and Technology of Integrated Systems in Nanoscale Era*, pp. 102–107, Sep. 2007.

[16] K. Wahid, M. Martuza, M. Das, and C. McCrosky, "Efficient Hardware Implementation of 8x8 Integer Cosine Transforms for Multiple Video Codecs," *Journal of Real-Time Image Processing*, Springer, in press, 2011.

[17] Guiying Liu, "An area-efficient IDCT architecture for multiple video standards," in *Proc. 2nd Int. Conf. ICISE*, pp. 3518–3522, Dec. 2010.

[18] C.P. Fan, and G.A. Su, "Efficient Low-Cost Sharing Design of Fast 1D Inverse Integer Transform Algorithms for H.264/AVC and VC-1," *IEEE Signal Process. Lett.*, vol. 15, pp. 926–9292, Dec. 2008.

[19] C. Fan and G. Su, "Fast Algorithm and Low-Cost Hardware-Sharing Design of Multiple Integer Transforms for VC-1," *IEEE Trans. Circuits and Syst. II*, vol. 56, pp. 788–792, Oct. 2009.

[20] Dajiang Zhou, et al., "A 1080p@60fps multi-standard video decoder chip designed for power and cost efficiency in a system perspective," in Symposium on VLSI Circuits, pp. 262–263, Jun. 2009.

[21] C. P. Fan and Y. L. Lin, "Implementations of Low Cost Hardware Sharing Architectures for Fast 8x8 and 4x4 Integer Transforms in H.264/AVC," *IEICE Trans. Fund. of Elect. Comm. and Compt.*, vol. 90, pp. 511–516, 2007.

[22] Y. C. Chao, et al., "An Efficient Architecture of Multiple 8x8 Transforms for H.264/AVC and VC-l Decoders," *Green Circuits and Syst.*, pp. 595–598, 2010.

[23] M. Martuza and K. Wahid, "A Cost Effective Implementation of 8x8 Transform of HEVC from H.264/AVC," *Proc. 25th IEEE CCECE*, in press, 2012.

[24] M. Martuza, C. McCrosky and K. Wahid, “A Fast Hybrid DCT Architecture Supporting H.264, VC-1, MPEG-2, AVS, and JPEG Codecs,” *Proc. 11th Int. Conf. ISSPA*, in press, 2012.

[25] M. Martuza and K. Wahid, “Low Cost Design of a Hybrid Architecture of Integer Inverse DCT for AVS, H.264, VC-1 and HEVC,” *VLSI Design*, Special issue on VLSI Circuits, Systems, and Architectures for Advanced Image and Video Compression Standards, Hindawi Publishing, 11 pages, in press, 2012.

[26] K. Wang, et al., “A Reconfigurable Multi-Transform VLSI Architecture Supporting Video Codec Design,” IEEE Trans. Circuits and Syst. II, Exp. Briefs, vol. 58, no. 7, pp. 432–436, July 2011.

[27] ITU-T Rec. H.264/ISO/IEC 14496-10 AVC, 2003.

[28] Standard for Television: VC-1 Compressed Video Bitstream Format and Decoding Process, SMPTE 421M 2006.

[29] G. J. Sullivan and J.-R. Ohm, “Recent developments in standardization of high efficiency video coding (HEVC),” in *SPIE Applications of Digital Image Processing XXXIII*, vol. 7798, Aug. 2010.

[30] W. Gao, et al., “AVS Video Coding Standard,” Intelligent Multimedia Communication: Techniques and Applications, vol. SCI-280, pp. 125–166, 2010.

[31] K. Ugur, et al., “High Performance, Low Complexity Video Coding and the Emerging HEVC Standard,” in *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 20, no. 12, pp. 1688–1697, Dec. 2010.

[32] GB/T 20090.1 Information technology - Advanced coding of audio and video - Part 1: System, Chinese AVS standard.

[33] S. David and M. Giovanni, Handbook of Data Compression, 5th Ed., 2010.

[34] *Discrete Cosine Transform* [Online]. Available: http://en.wikipedia.org/wiki/ Discrete_cosine_transform. Accessed on: April 25, 2012.

[35] X. Shao and S. G. Johnson, “Type-II/III DCT/DST algorithms with reduced number of arithmetic operations,” *Signal Processing*, vol. 88, no. 6, pp. 1553–1564, 2008.

[36] ISO/IEC 14 496-10 and ITU-T Rec. H.264, Advance Video Coding, 2003.

[37] S. Kwon, A. Tamhankar, K.R. Rao, “Overview of H.264/MPEG-4 part 10,” in *J. Visual Communication and Image Representation*, Special issue on Emerging H.264/AVC video coding standard, vol. 17, pp. 186–216, Apr. 2006.

[38] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, “Low complexity transform and quantization in H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 598–603, Jul. 2003.

[39] S. Gordon, D. Marple, and T. Wiegand, "Simplified use of 8x8 transforms — Updated proposal and results," in *Proc. JVT-K028, 11th Meeting*, Munich, Germany, Mar. 2004.

[40] *VC-1* [Online]. Available: http://en.wikipedia.org/wiki/VC-1. Accessed on: April 25, 2012.

[41] International Standards Organization, ISO/IEC 13818-7, MPEG-2 Standards Document.

[42] W. Pennebaker, J. Mitchell, JPEG Still Image Data Compression Standard, van Nostrand Reinhold, New York, 1993.

[43] *Tektronics Picture Quality Analyzer PQA 300* [Online]. Available: http://www.tek.com/datasheet/picture-quality-analysis-system-2. Accessed on: April 27, 2012.

[44] Y. Huh, K. Panusopone, K.R. Rao, "Variable block size coding of images with hybrid quantization," *IEEE Trans. Circuits Systems Video Technol.*, vol. 6, pp. 679–685, Dec. 1996.

[45] N. Ranganthan, S.G. Romaniuk, K.R. Namuduri, "A lossless image compression algorithm using variable size block segmentation," *IEEE Trans. Image Proccess.*, vol. 4, no. 10, pp. 1396–1407, 1995.

[46] J. Ribas-Corbera, D.L. Neuhoff, "Optimizing block size in motion compensation," *J. Electron. Imaging*, vol. 1, pp. 155–165, Jan. 1998.

[47] S. Srinivasan, et al., "Fast Video Codec Transform Implementations," U.S. Patent 20050256916, Nov. 2005.

[48] Y. Li, Y. He, and S. L. Mei, "A highly parallel joint VLSI architecture for transforms in H.264/AVC," in J. *Signal Process. Syst.*, vol. 50, no. 1, pp. 19–32, Jan. 2008.

[49] W. Gao, et al., "AVS Video Coding Standard," in *Studies in Computational Intelligence*, Intelligent Multimedia Communication: Techniques and Applications, Springer- Verlag Berlin Heidelberg, pp. 125–166, 2010.

[50] C. Zhang, et al., "The Techniques of Pre-scaled Integer Transform," in *Int. Symp. Circuits Syst.*, vol. 1, pp. 316–319, 2005.

[51] *JCTVC HM* [Online]. Available: http://hevc.kw.bbc.co.uk/trac /browser/tags/0.9. Accessed on: April 27, 2012.

[52] A. Fuldseth, G. Bjøntegaard, and M. Budagavi, "CE10: Core transform design for HEVC," *JCTVC-G495*, Geneva, Switzerland, Nov. 2011.

[53] Chi-Cheng Ju, et al., "A Full-HD 60fps AVS/H.264/VC-1/MPEG-2 Video Decoder for Digital Home Applications," in *Proc. Int. Symp. VLSI-DAT*, pp. 1–4, April, 2011.

[54] K. R. Rao, D. N. Kim, "Current Video Coding Standards: H.264/AVC, Dirac, AVS China and VC-1," *42nd South Eastern Symposium on System Theory*, University of Texas, TX, USA, March 2010.

[55] G. A. Su and C. P. Fan, "Low-cost hardware-sharing architecture of fast 1-D inverse tranforms for H.264/AVC and AVS applications," in *IEEE Trans. Circuits and Syst. II, Exp. Briefs*, vol. 45, no. 12, pp. 1249–12153, Dec. 2008.

[56] A. Vetro, et al., "Video Transcoding Architectures and Techniques: An Overview," *IEEE Signal Process. Magz.*, vol. 20, pp. 18–29, Mar. 2003.

[57] *CIPR Sequence of 'Trevor'* [Online]. Available: http://www.cipr.rpi.edu/resource /sequences/sequences/sequence01/sequence01_trevor_gray.zip. Accessed on: May 9, 2012.

[58] G. Strang and T. Nguyen, Wavelets and Filter Banks, Cambridge Press, 1996.

[59] M. Islam, "Hardware Implementation of Daubechies Wavelet Transforms using Folded AIQ Mapping," M.S. thesis, Dept. of ECE, Univ. of Sask., Saskatoon, SK, 2010.

[60] V. S. Rao, et al., "Discrete Cosine Transform Vs Discrete Wavelet Transform: An Objective Comparison of Image Compression Techniques for JPEG Encoder," *Int. J. Advanced Engineering & Applications*, pp. 87–90, Jan. 2010.

[61] J. Stoer and R. Bulirsch, Introduction to Numerical Analysis, 3rd Ed. Berlin, New York: Springer-Verlag, 2002, p. 619.

[62] T. Davis. *Sparse Matrix* [Online]. Available: http://mathworld.wolfram.com /SparseMatrix.html. Accessed on: June 13, 2012.

[63] G. Breed, "A Comparison of RFIC Fabrication Technologies," *J. High Frequency Electronics*, pp. 52–53, Mar. 2006.

[64] *Xilinx® Virtex-4 Family Overview: Product Specification* [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf. Accessed on: June 13, 2012.